Exhibit V

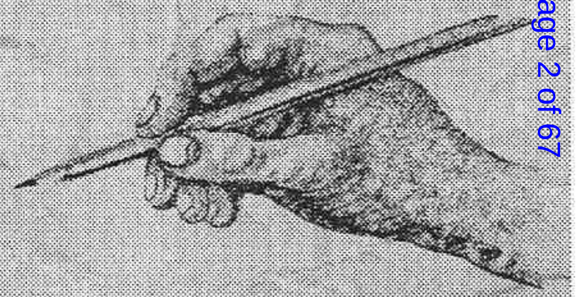
To

Joint Claim Chart

Covers Windows 95 & Windows NT™!

The Windows® Interface Guidelines for Software Design

MICROSOFT® WINDOWS™



PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Cataloging-in-Publication Data
The Windows interface guidelines for software design.
p. cm.
Includes index.
ISBN 1-55615-679-0
1. Microsoft Windows (Computer file) 2. User interfaces (Computer systems) 3. Computer software--Development. I. Microsoft Corporation.
QA76.76.W56W553 1995
005.265--dc20 95-330
CIP

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9 QEQE 0 9 8 7 6 5

Distributed to the book trade in Canada by Macmillan of Canada, a division of Canada Publishing Corporation.

A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office. Or contact Microsoft Press International directly at fax (206) 936-7329.

Information in this document is subject to change without notice and does not represent a commitment on the part of Microsoft Corporation. Companies, names, and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property rights.

Adobe, PostScript, and TIFF are trademarks of Adobe Systems, Inc. Apple and TrueType are registered trademarks of Apple Computer, Inc. Borland and Quattro are registered trademarks of Borland International, Inc. Frutiger is a registered trademark of Eltra Corporation. HP and LaserJet are registered trademarks of Hewlett-Packard Company. Backup was developed for Microsoft by Colorado Memory Systems, Inc., a division of Hewlett-Packard Company. HyperTerminal is a trademark of Hilgraeve, Inc. 1-2-3 and Lotus are registered trademarks of Lotus Development Corporation. Microsoft, Microsoft Press, Microsoft Press logo, MS, MS-DOS, PowerPoint, Visual Basic, Windows, Windows logo, and XENIX are registered trademarks and Windows NT is a trademark of Microsoft Corporation. Arial, Bodoni, Swing, and Times New Roman are registered trademarks of The Monotype Corporation PLC. Paintbrush is a trademark of Wordstar Atlanta Technology Center.

V

visual editing *See* OLE visual editing.

W

well control A control that is used to display color or pattern choices, typically used like an option button.

white space The background area of a window. (The color need not literally be white.)

window A standard Windows object that displays information. A window is a separately controllable area of the screen that typically has a rectangular border. *See also* primary window and secondary window.

wizard A form of user assistance that automates a task through a dialog with the user.

wordwrap The convention where, as a user enters text, existing text is automatically moved from the end of a line to the next line.

workbook A window or task management technique that consists of a set of views that are organized like a tabbed notebook.

workspace A window or task management technique that consists of a container holding a set of objects, where the windows of the contained objects are constrained to a parent window. Similar to the multiple document interface, except that the windows displayed within the parent window are of objects that are also contained in the workspace.

writing tool A standard Windows pen interface control that supports text editing.

Z

Z order The layered relationship of a set of objects, such as windows, on the display screen.

3ibl

ieneral L

aecker, Ron
adings in H
ultidiscipl
aufmann, 198

ooks, Freder
ssays on Sof
ddison-Wesle

eckel, Paul. T
esign. New E

akoff, George
ve By. Chicag

aurel, Brenda.
terface Desig
ub. Co., 1990.

orman, Donal
ew York: Basi

orman, Donal
ser Centered
uman-Compu
lbaum Associ

open appearance The visual display of an object when the user opens the object into its own window.

operation A generic term that refers to the actions that can be done to or with an object.

option button A standard Windows control that allows a user to select from a fixed set of mutually exclusive choices (also referred to as a radio button). *Compare* check box.

option-set appearance The visual display for a control when its value is set.

outside-in activation A technique that requires a user to perform an explicit activation command to interact with the content of an OLE embedded object. *Compare* inside-out activation.

P

package An OLE encapsulation of a file so that it can be embedded in an OLE container.

palette window A modeless secondary window that displays a toolbar or other choices, such as colors or patterns. *Compare* dialog box and message box. *See also* property sheet.

pane One of the separate areas in a split window.

parent window A primary window that provides window management for a set of child windows. *See also* child window and multiple document interface.

pen An input device that consists of a pen-shaped stylus that a user employs to interact with a computer.

persistence The principle that the state of an object is automatically preserved.

point (v.) To position the pointer over a particular object and location. (n.) A unit of measurement for type (1 point equals approximately 1/72 inch).

pointer A graphic image displayed on the screen that indicates the location of a pointing device (also referred to as a cursor).

pop-up menu A menu that is displayed at the location of a selected object (also referred to as a context menu or shortcut menu). The menu contains commands that are contextually relevant to the selection.

pop-up window A secondary window with no title bar that is displayed next to an object; it provides contextual information about that object.

portrait An orientation where the long dimension of a rectangular area (for example, screen or paper) is vertical.

press To press and release a keyboard key or to touch the tip of a pen to the screen. *See also* click.

pressed appearance The visual display for an object, such as a control, when it is being pressed.

primary window The window in which the main interaction takes place. *See also* secondary window and window.

progress indicator Any form of feedback that provides the user with information about the state of a process.

progress indicator control A standard Windows control that displays the percentage of completion of a particular process as a graphical bar.

Exhibit W

To

Joint Claim Chart

US005760773A

United States Patent [19]

Berman et al.

[11] **Patent Number: 5,760,773**

[45] **Date of Patent: Jun. 2, 1998**

[54] **METHODS AND APPARATUS FOR INTERACTING WITH DATA OBJECTS USING ACTION HANDLES**

[75] Inventors: **Eric Robert Berman**, Redmond; **Edward Low Mills**, Bellevue; **Michael Hinkley Van Kleeck**, Duvall; **Vinayak A. Bhalerao**, Redmond, all of Wash.

[73] Assignee: **Microsoft Corporation**, Redmond, Wash.

[21] Appl. No.: **369,688**

[22] Filed: **Jan. 6, 1995**

[51] **Int. Cl.^{6}** **G06F 3/00**

[52] **U.S. Cl.** **345/347**; 345/121

[58] **Field of Search** 395/155, 159, 395/156, 161, 146, 133, 347, 339, 348, 354, 352, 334, 335; 345/145, 146, 121, 347, 339, 348, 354, 352, 334, 335, 433

[56] **References Cited**

U.S. PATENT DOCUMENTS

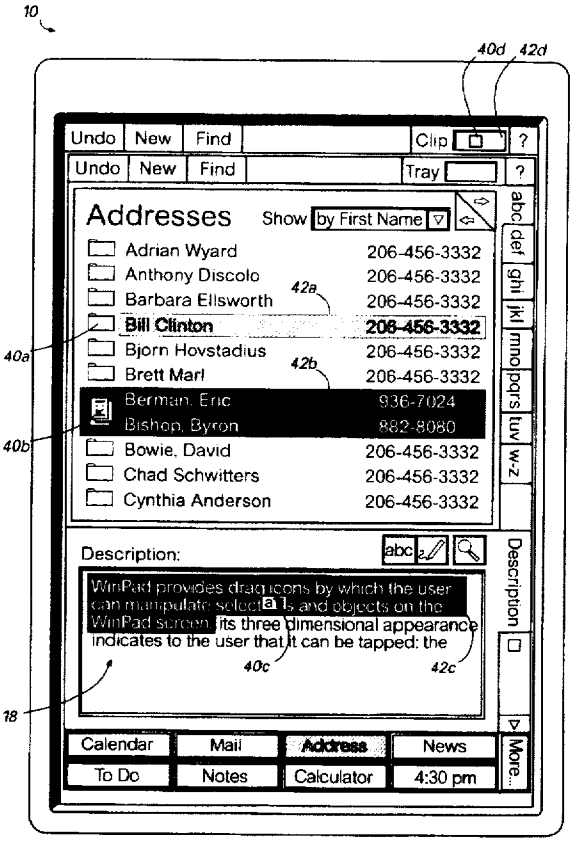
5,196,838	3/1993	Meier et al.	345/118
5,345,543	9/1994	Capps et al.	345/437
5,347,295	9/1994	Agulnick et al.	345/156
5,367,453	11/1994	Capps et al.	707/531
5,390,281	2/1995	Luciw et al.	395/12
5,396,590	3/1995	Kreegar	345/347
5,450,539	9/1995	Ruben	345/354
5,513,309	4/1996	Meier et al.	345/339
5,515,496	5/1996	Kaehler et al.	345/334
5,570,281	10/1996	Berry	364/146

Primary Examiner—Matthew M. Kim
Assistant Examiner—Crescelle N. dela Torre
Attorney, Agent, or Firm—Jones & Askew, LLP

[57] **ABSTRACT**

A central processing unit (CPU) is coupled to a computer display for displaying graphic and other information. The CPU is further coupled to a pointer control device such as a pen, stylus or mouse that permits the user to selectively position a pointer and activate an action handle on the display associated with a data object. Activation of the action handle signals the CPU of selections associated with the data object. Tapping or clicking on the action handle causes display of a context menu containing at least one command that may be invoked with respect to the data object. Dragging the action handle indicates movement or dragging of the action handle and the corresponding data object for an operation associated with dragging such as drag-and-drop. The use of action handles in association with data objects eliminates ambiguity between actions intended to create digital ink with respect to a pen-based computer system and actions intended to invoke a command or other interaction with the computer system.

30 Claims, 12 Drawing Sheets



Addresses Show by First Name ▽

Adrian Wyard	206-456-3332
Anthony Discolo	206-456-3332
Barbara Ellsworth	206-456-3332
Bill Clinton	**206-456-3332**
Bjorn Hovstadius	206-456-3332
Brett Marl	206-456-3332
Berman, Eric	936-7024
Bishop, Byron	882-8080
Bowie, David	206-456-3332
Chad Schwitters	206-456-3332
Cynthia Anderson	206-456-3332

WinPad provides drag icons by which the user can manipulate selectals and objects on the WinPad screen. its three dimensional appearance indicates to the user that it can be tapped: the

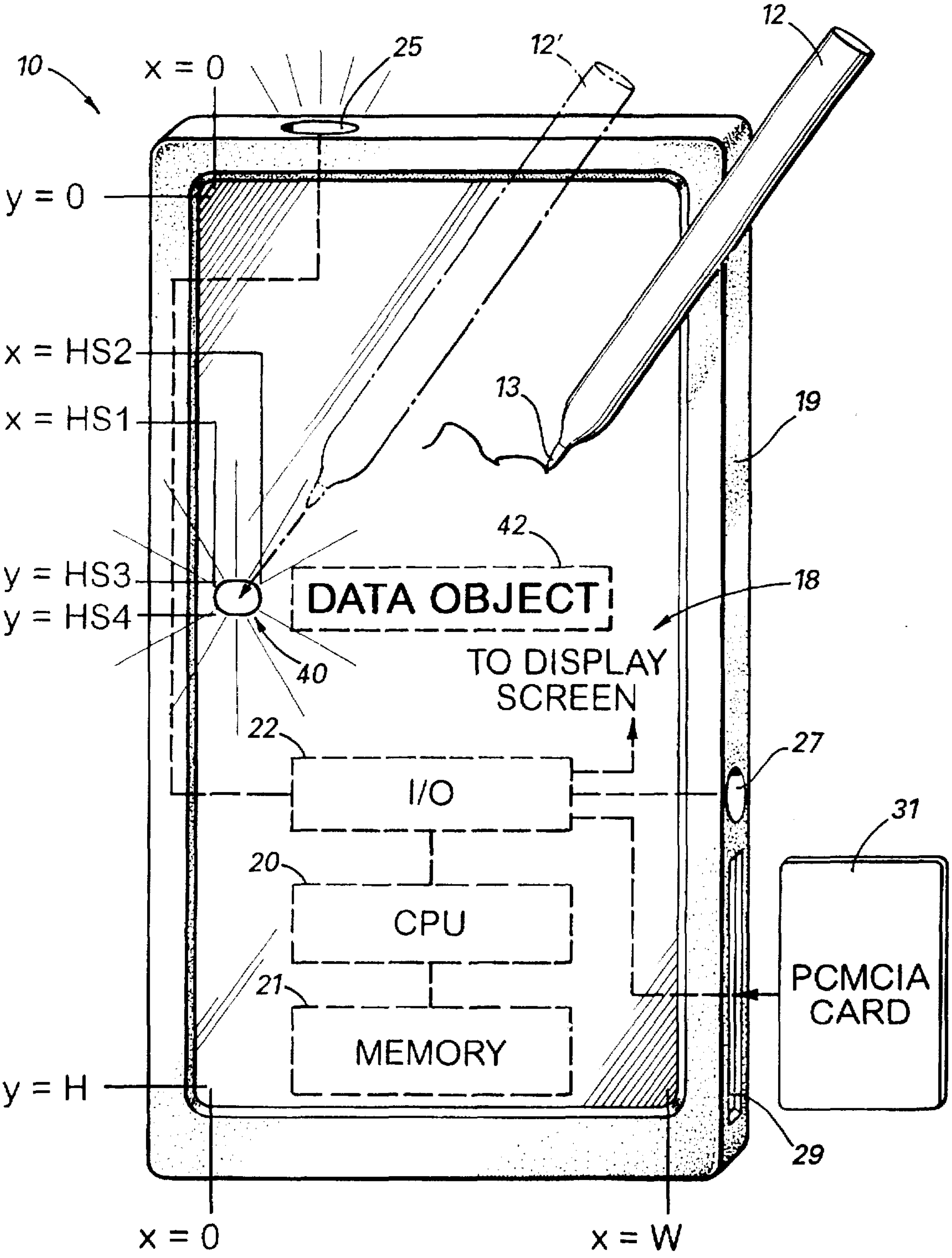
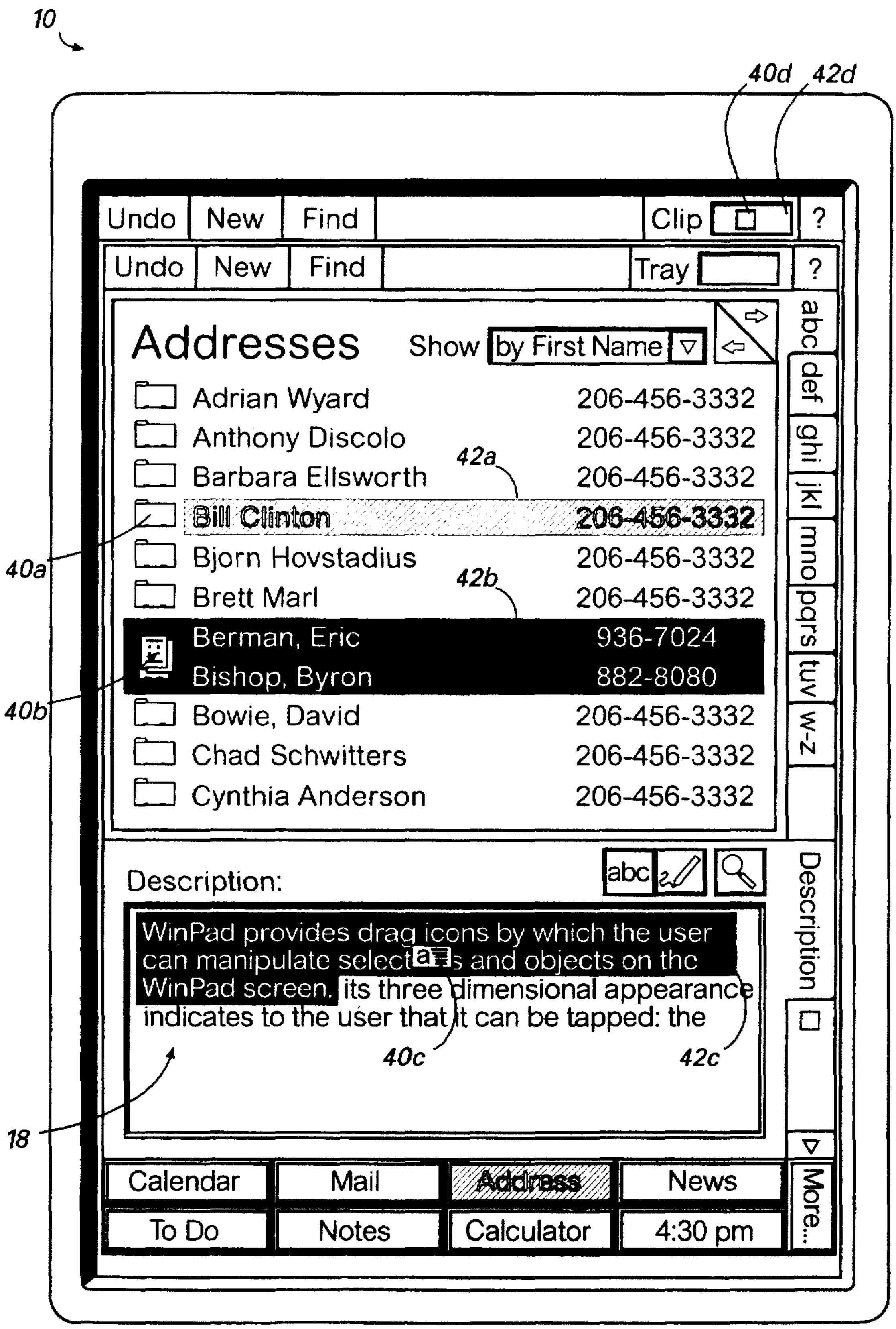

10
x = 0
25
12'
12
y = 0
x = HS2
x = HS1
13
19
42
y = HS3
y = HS4
DATA OBJECT
18
40
TO DISPLAY SCREEN
22
27
I/O
31
20
CPU
PCMCIA CARD
21
MEMORY
y = H
29
x = 0
x = W

FIG. 1



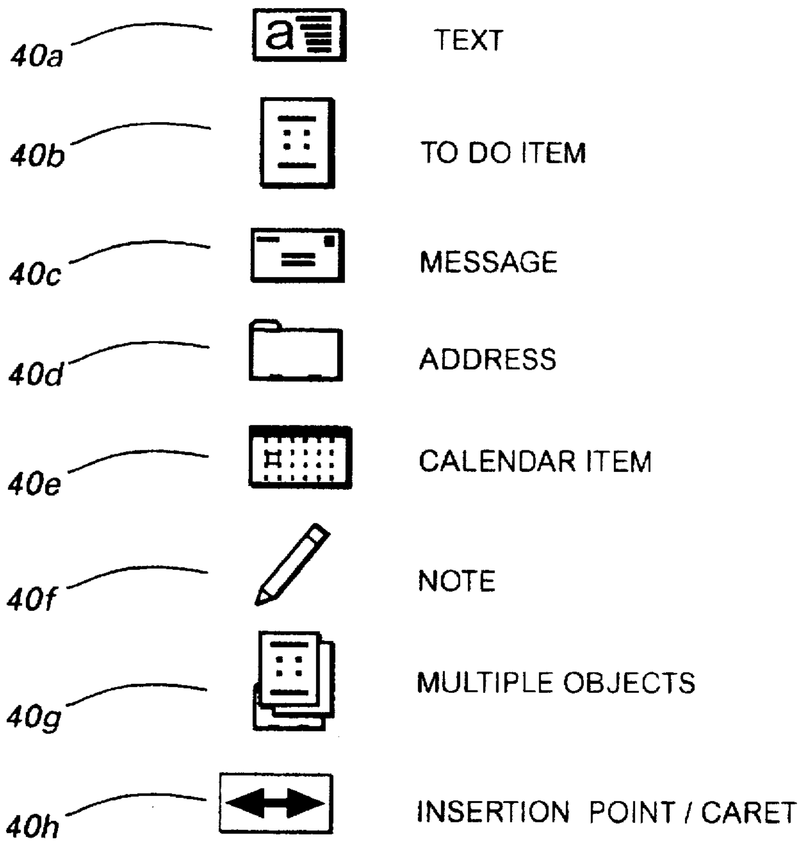
Addresses

Adrian Wyard	206-456-3332
Anthony Discolo	206-456-3332
Barbara Ellsworth	206-456-3332
Bill Clinton	206-456-3332
Bjorn Hovstadius	206-456-3332
Brett Marl	206-456-3332
Berman, Eric	936-7024
Bishop, Byron	882-8080
Bowie, David	206-456-3332
Chad Schwitters	206-456-3332
Cynthia Anderson	206-456-3332

Description:

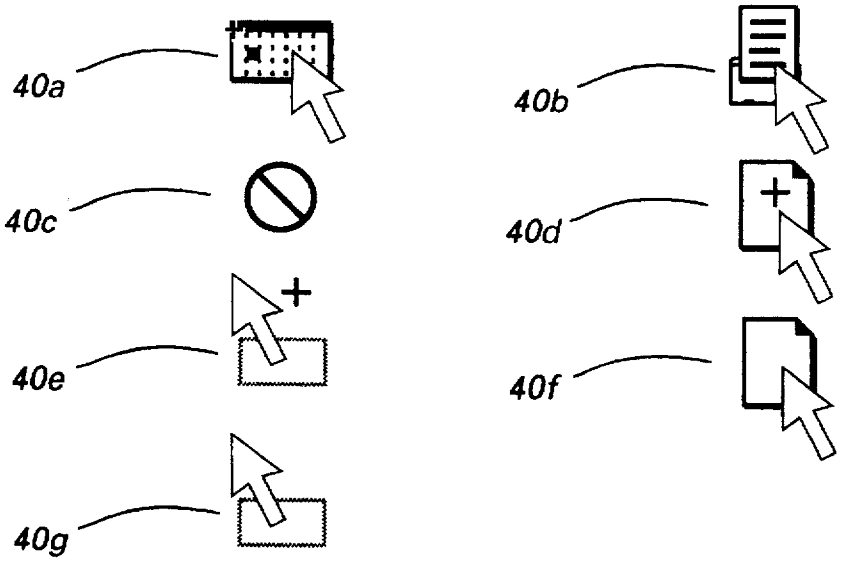
WinPad provides drag icons by which the user can manipulate select▤s and objects on thc WinPad screen. its three dimensional appearance indicates to the user that it can be tapped: the

FIG.2



EXEMPLARY STATIC ACTION HANDLES

FIG.3



EXEMPLARY DRAG ICONS
(DYNAMIC OR STATUS INDICATING ACTION HANDLES

FIG.4

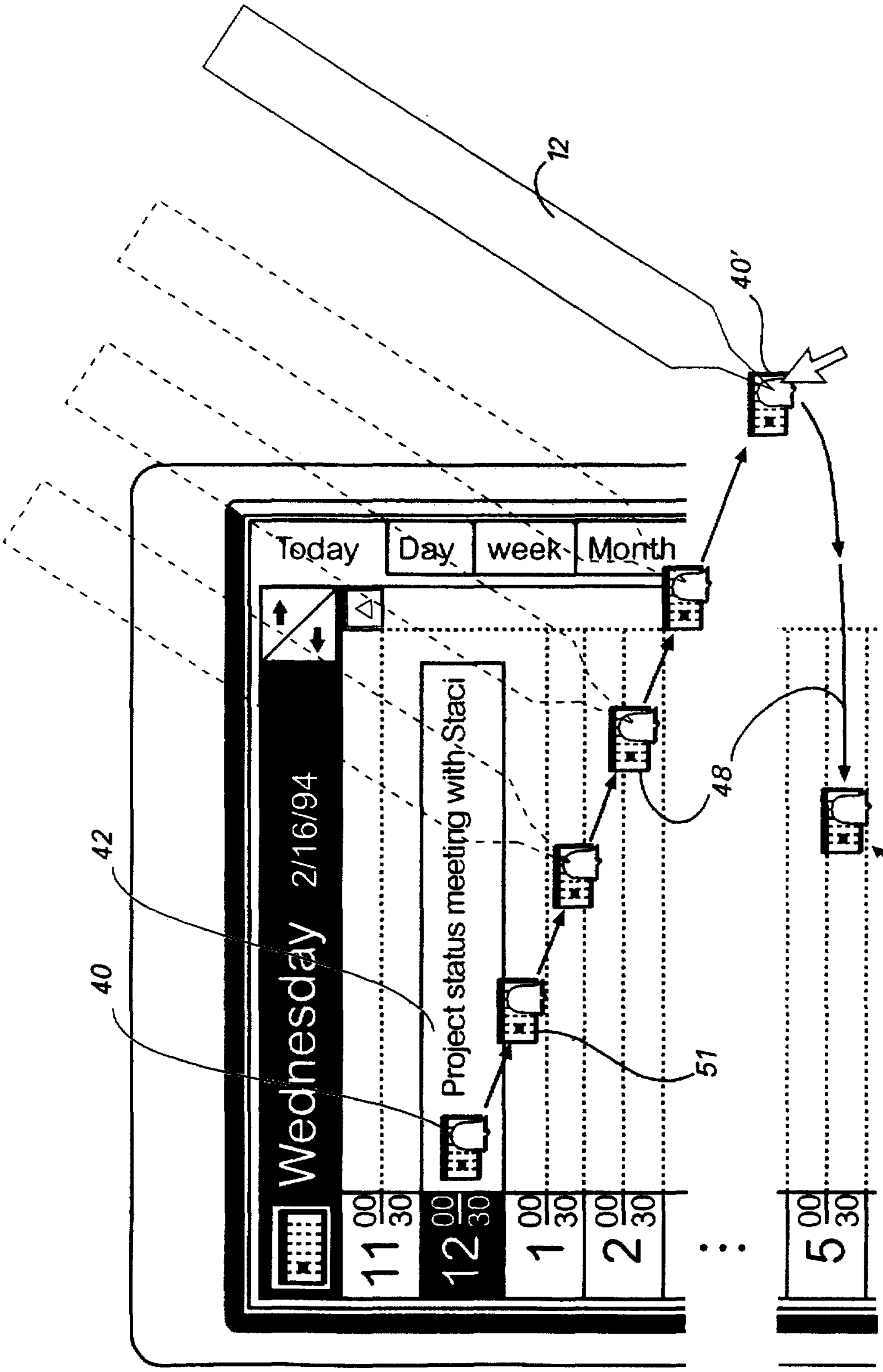
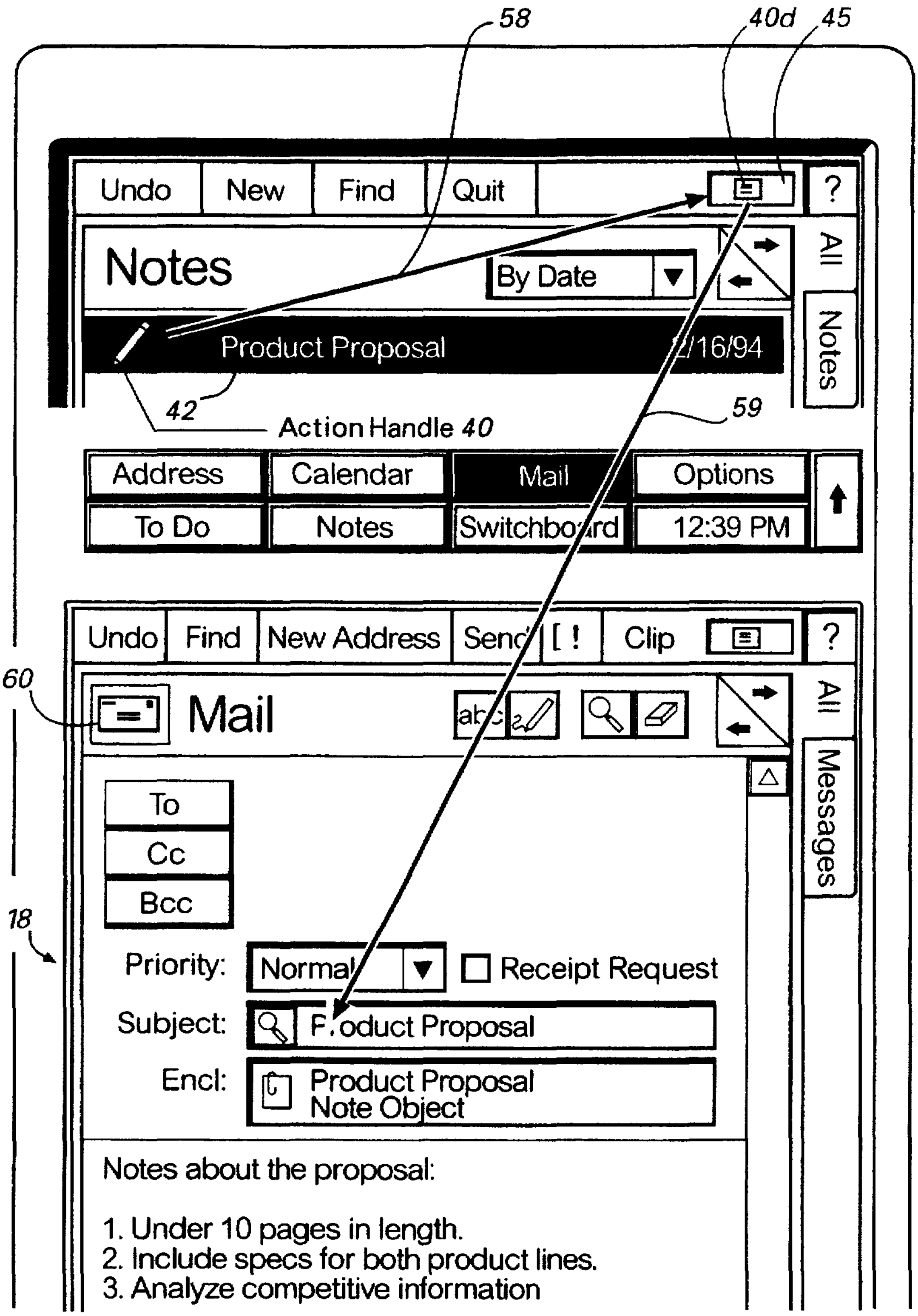

12
40'
Today
Day
week
Month
Wednesday
2/16/94
Project status meeting with Staci
42
40
51
48
53
11 00 30
12 00 30
1 00 30
2 00 30
...
5 00 30

FIG.5



Priority: Normal ▼ □ Receipt Request

Subject: Product Proposal

Encl: Product Proposal Note Object

Notes about the proposal:

1. Under 10 pages in length.
2. Include specs for both product lines.
3. Analyze competitive information

FIG.6

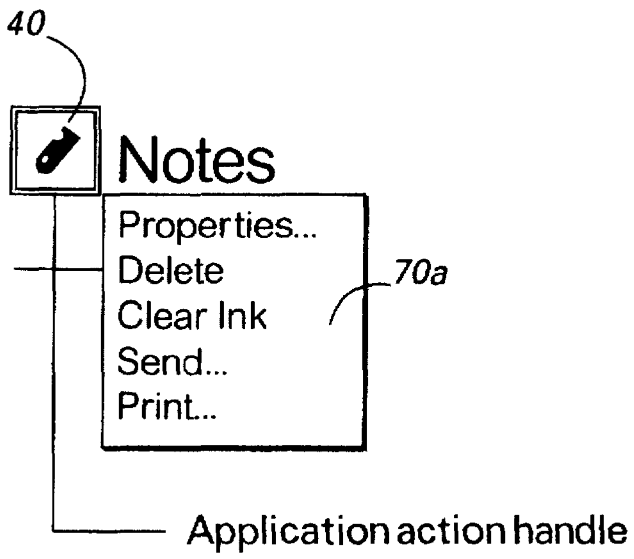

40
Notes
Properties...
Delete
Clear Ink
Send...
Print...
70a
Application action handle

FIG.7A

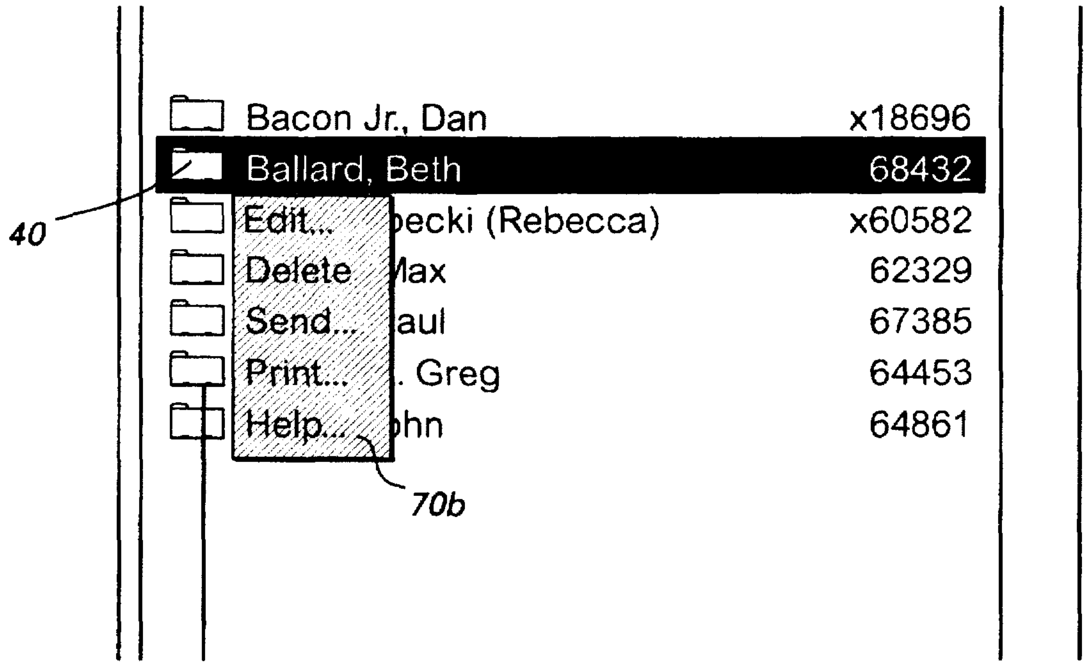
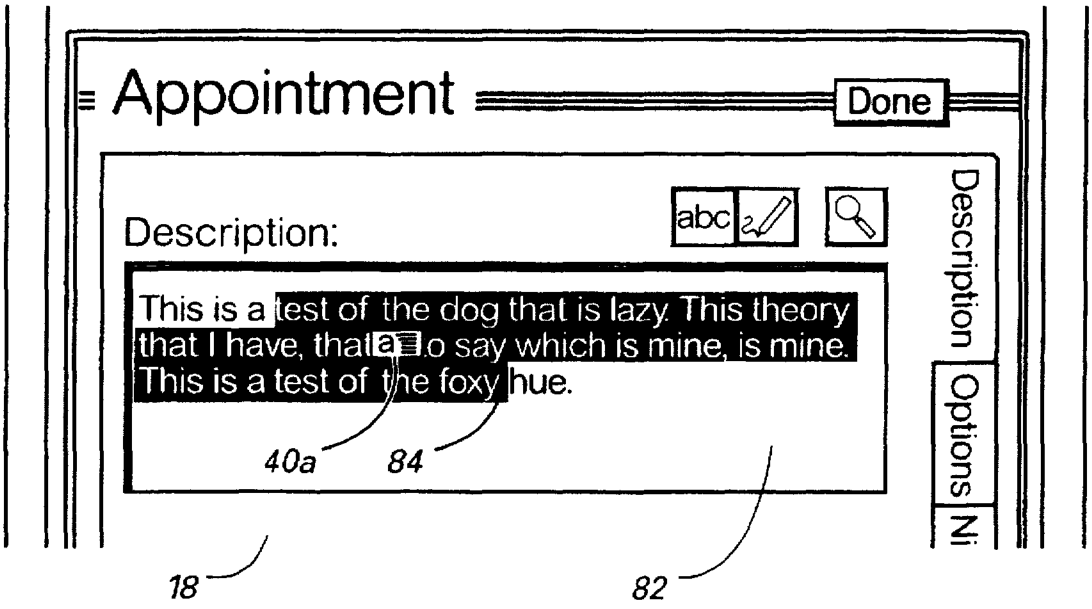
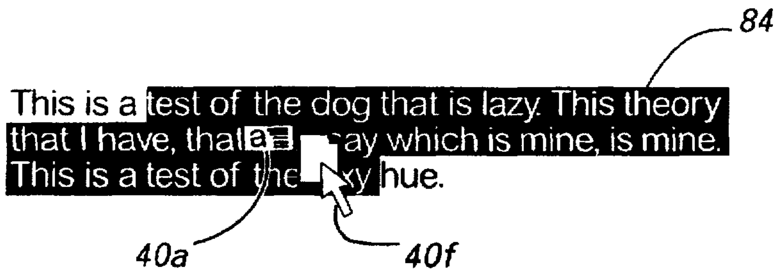

Bacon Jr., Dan x18696
Ballard, Beth 68432
Edit... ecki (Rebecca) x60582
Delete lax 62329
Send... aul 67385
Print... . Greg 64453
Help... hn 64861
40
70b

FIG.7B



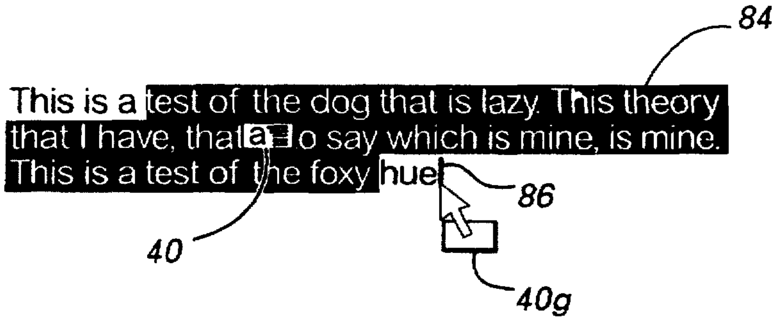
This is a test of the dog that is lazy. This theory that I have, that a o say which is mine, is mine. This is a test of the foxy hue.

FIG. 8A



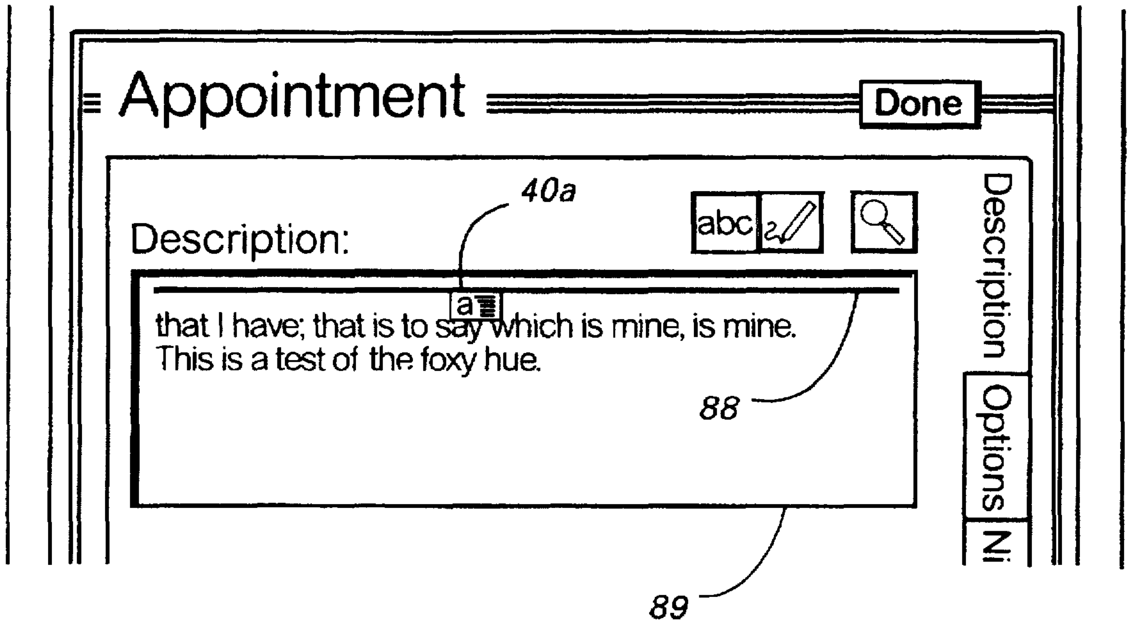
This is a test of the dog that is lazy. This theory that I have, that a ay which is mine, is mine. This is a test of the xy hue.

FIG. 8B



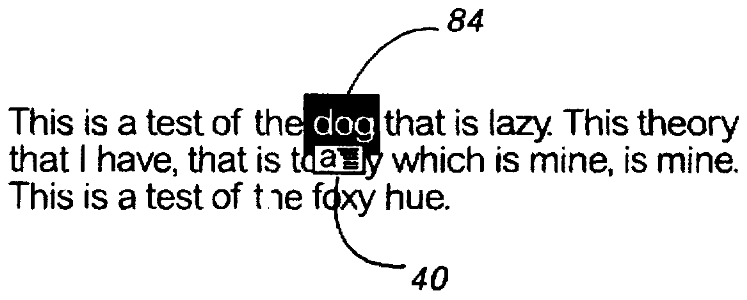
This is a test of the dog that is lazy. This theory that I have, that a o say which is mine, is mine. This is a test of the foxy hue

FIG. 8C



that I have; that is to say which is mine, is mine.
This is a test of the foxy hue.

FIG.9



This is a test of the dog that is lazy. This theory
that I have, that is to a y which is mine, is mine.
This is a test of the foxy hue.

FIG.10

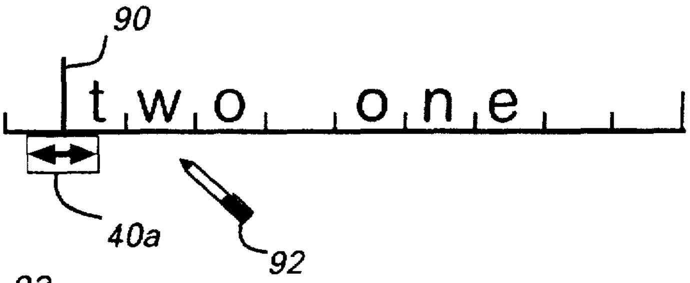


FIG.11A

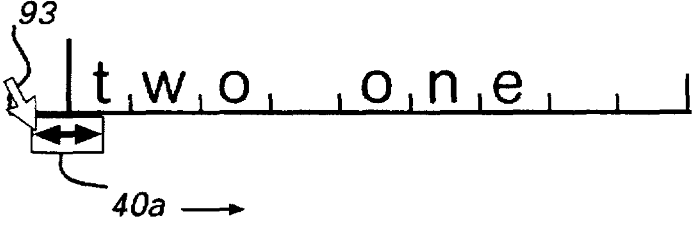


FIG.11B

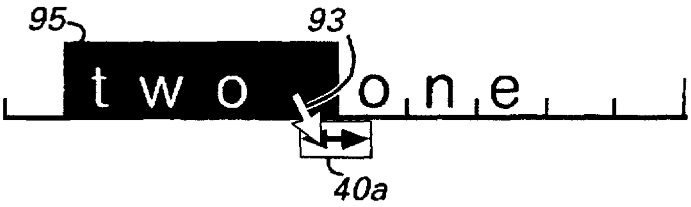


FIG.11C

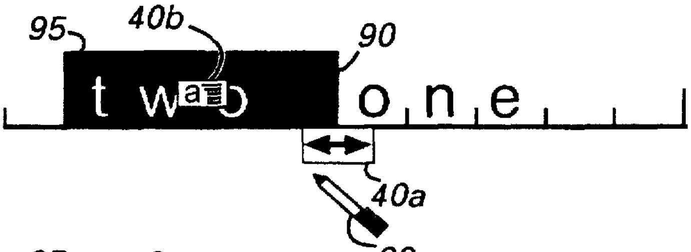


FIG.11D

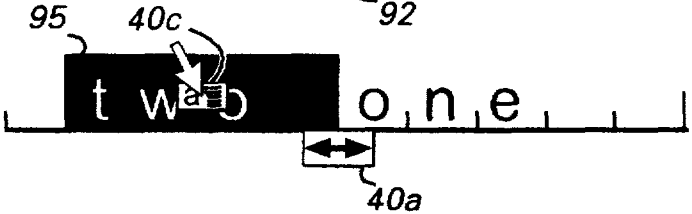


FIG.11E

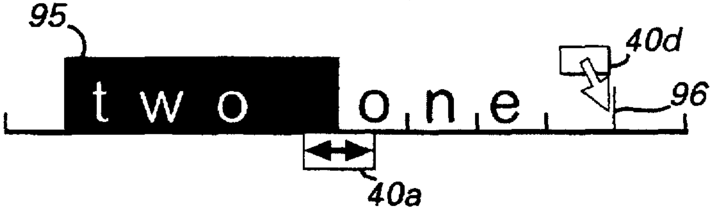


FIG.11F

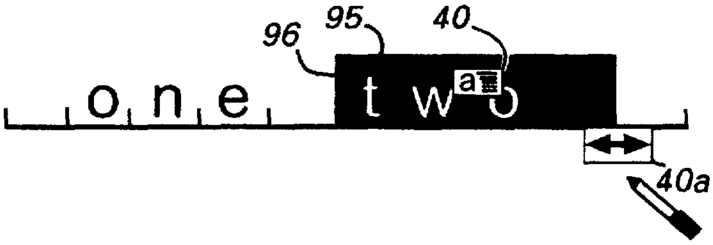
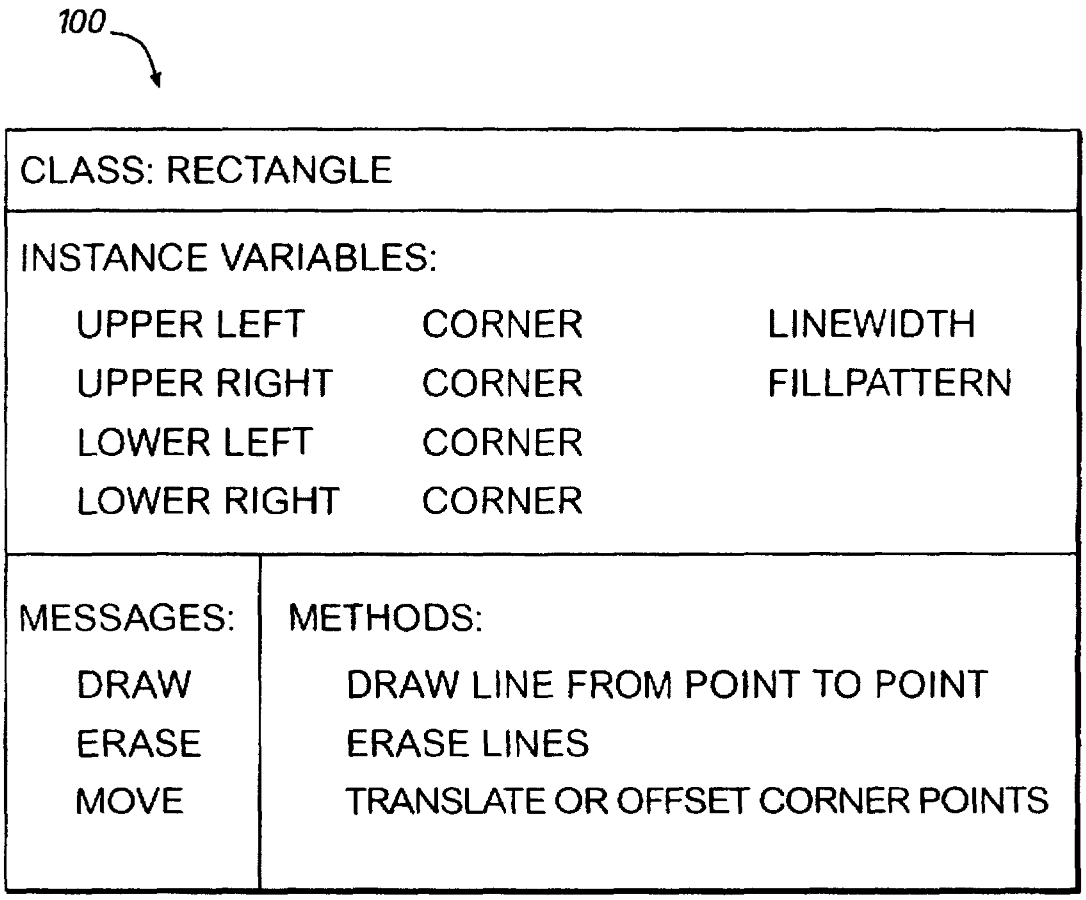


FIG.11G



CLASS: RECTANGLE

INSTANCE VARIABLES:

UPPER LEFT CORNER
UPPER RIGHT CORNER
LOWER LEFT CORNER
LOWER RIGHT CORNER
LINEWIDTH
FILLPATTERN

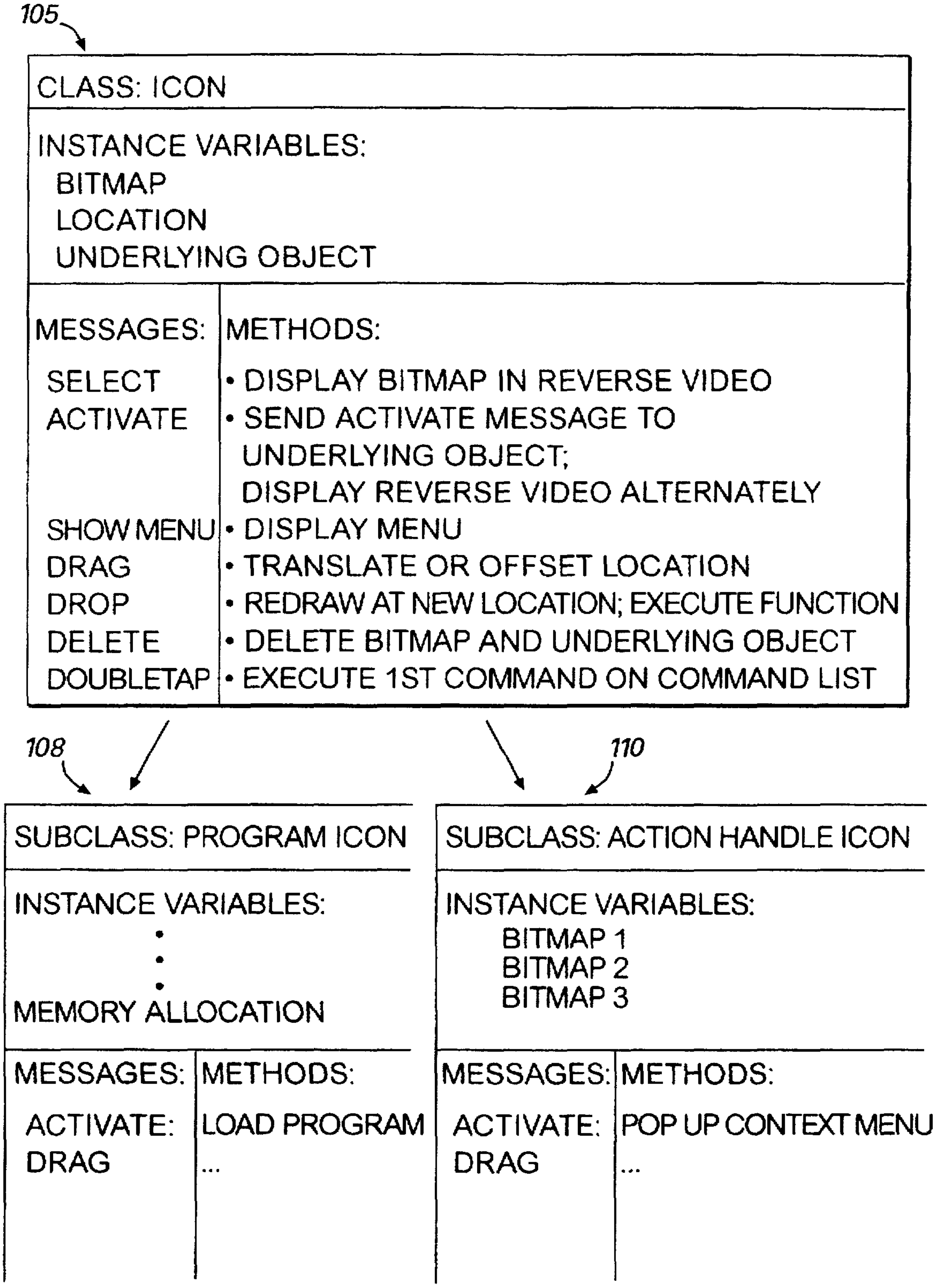
MESSAGES:

DRAW
ERASE
MOVE

METHODS:

DRAW LINE FROM POINT TO POINT
ERASE LINES
TRANSLATE OR OFFSET CORNER POINTS

FIG.12



CLASS: ICON

INSTANCE VARIABLES:
BITMAP
LOCATION
UNDERLYING OBJECT

MESSAGES:	METHODS:
SELECT	• DISPLAY BITMAP IN REVERSE VIDEO
ACTIVATE	• SEND ACTIVATE MESSAGE TO UNDERLYING OBJECT; DISPLAY REVERSE VIDEO ALTERNATELY
SHOW MENU	• DISPLAY MENU
DRAG	• TRANSLATE OR OFFSET LOCATION
DROP	• REDRAW AT NEW LOCATION; EXECUTE FUNCTION
DELETE	• DELETE BITMAP AND UNDERLYING OBJECT
DOUBLETAP	• EXECUTE 1ST COMMAND ON COMMAND LIST

SUBCLASS: PROGRAM ICON

INSTANCE VARIABLES:
•
•
•
MEMORY ALLOCATION

MESSAGES:	METHODS:
ACTIVATE:	LOAD PROGRAM
DRAG	...

SUBCLASS: ACTION HANDLE ICON

INSTANCE VARIABLES:
BITMAP 1
BITMAP 2
BITMAP 3

MESSAGES:	METHODS:
ACTIVATE:	POP UP CONTEXT MENU
DRAG	...

FIG.13

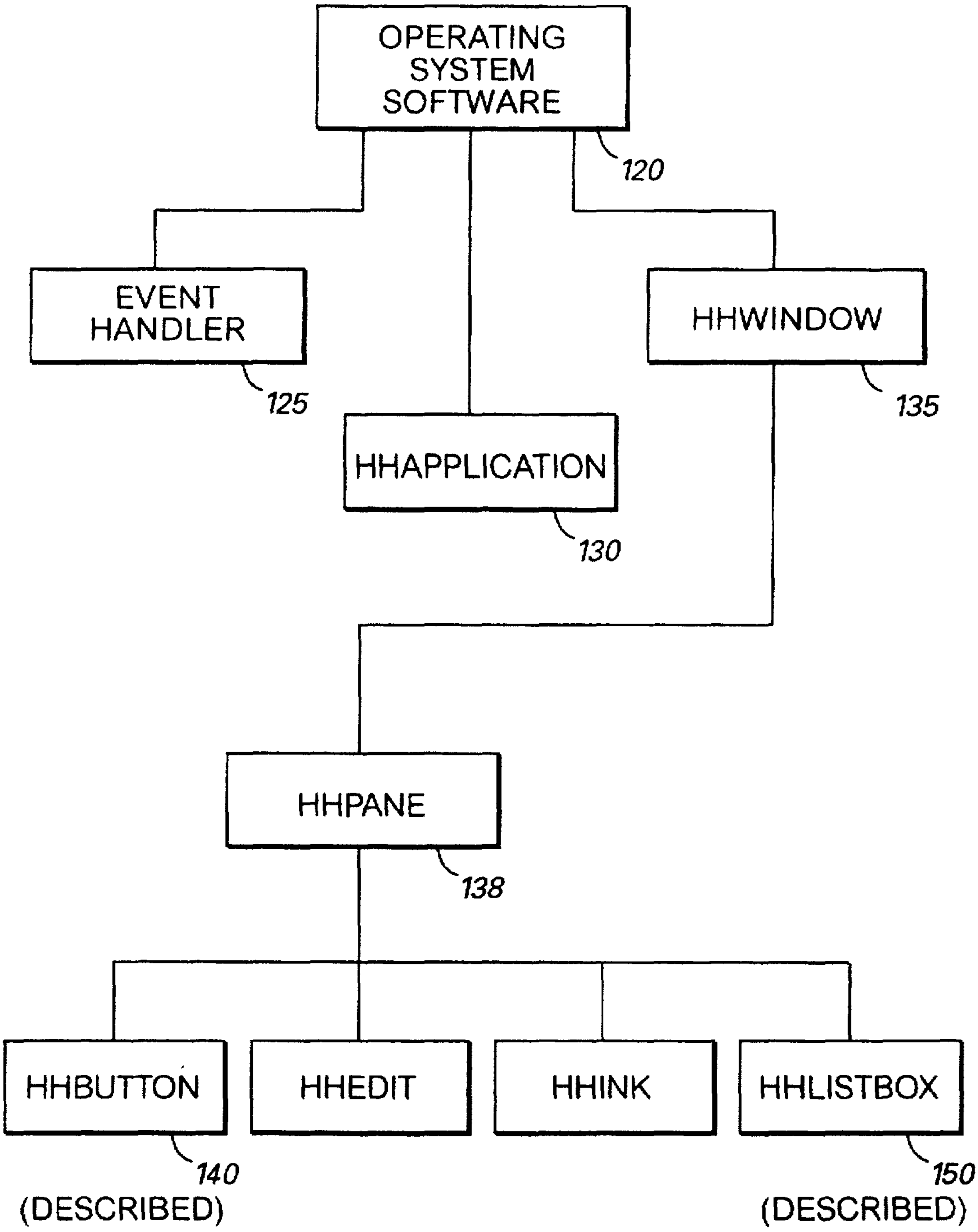

OPERATING SYSTEM SOFTWARE
120
EVENT HANDLER
125
HHWINDOW
135
HHAPPLICATION
130
HHPANE
138
HHBUTTON
140
(DESCRIBED)
HHEDIT
HHINK
HHLISTBOX
150
(DESCRIBED)

FIG. 14

METHODS AND APPARATUS FOR INTERACTING WITH DATA OBJECTS USING ACTION HANDLES

TECHNICAL FIELD

The present invention relates generally to apparatus and methods for displaying and manipulating information on a computer display screen, and more particularly relates to a computer controlled display system for displaying action handles or icons associated with varying types of data objects on a display screen, and for allowing user interaction with selected data objects by way of the action handles or icons.

BACKGROUND OF THE INVENTION

Present day computer systems often employ "object-oriented" displays with icons, pictures, text, pop-up or drop-down menus, dialog boxes, and other graphical items or objects in what is known as a "Graphical User Interface" or "GUI". In such systems, various combinations of graphical items or objects are utilized to convey information to users via a cathode ray tube (CRT) monitor or display. Users generally interact with the computer system by activating graphical items or objects on the display. The graphical items or objects are often activated by pointing at them with a mouse, pen, or other pointing device, and clicking or tapping a button.

Object oriented display systems often utilize small graphical items or objects called "icons", which symbolically indicate a type of operation the computer system will execute if the icon is selected by a user interface device such as a mouse, pen, or keyboard. These icons are frequently disposed within a window on a display. Icons are often used to delete or copy information, move information from one window to another, cause operations to the data in the window, and the like. An example of a computer graphical user interface that uses icons is the Microsoft WINDOWS™ operating system manufactured by the assignee of the present invention, Microsoft Corporation.

New generations of computer systems that use a pen or stylus as a user input device are being introduced. Some of the new pen-based computers are small, handheld devices with a display screen but no keyboard. Such devices are sometimes called "Personal Digital Assistants", "PDAs", or "portable pen pads". With such devices, a user interacts with a display screen like a sheet of paper, using an ink-less pen or stylus to draw pictures, to enter text via handwriting or printing, and to interact with the computer system. The computer interprets the motion of the pen or stylus relative to the display screen, and often displays lines or markings on the screen that track the movement of the pen. Markings on the screen displayed as a result of movement of the stylus are often called "digital ink".

The pen or stylus in a pen-based computing system may be considered a "direct pointing device", that is, a pointing device consisting of a display and an indicator such as a finger, stylus, or other object. The pointing-device actions of "pointing" and "tapping"—which are analogous to "clicking" with a mouse—are achieved by moving the indicator on or near the display screen and touching the display screen, respectively. With a mouse or trackball type pointing device, there is an indirection. The user moves some physical device (positioned somewhere away from the screen) in order to move an artificial pointer (i.e. a cursor) on the display screen. Without the cursor, there is no way for the user to "see where you are" on the screen. With a direct pointing device, the "where you are" is represented by the indicator itself, and, optionally, by a cursor.

Because there is no keyboard or mouse, pen-based computers have tighter restrictions on what a user can and cannot do. Common user interfaces such as those is found in the Microsoft WINDOWS™ operating system and other GUI-based computer systems typically rely on a keyboard and a pointing device with buttons such as a mouse or trackball for user interaction. The user points to a location on the screen where an action is desired, and invokes a command by pressing a button on the pointing device or entering a keyboard command. For example, to insert a space between words it is generally assumed the user can simply select a location for insertion of the space by using the mouse to position a cursor at the point of insertion and hit the space bar. Consequently, many present-day application software programs do not offer an "insert space" command for text editing.

Some known computer application programs add commands to existing drop-down menu bars. However, this method is not suitable for handheld pen-based applications since most menu bars are impracticably large on devices with a small screen.

Therefore, systems that are required to work with a pen or stylus must find alternative methods to accomplish such actions.

In pen-based computers, it is difficult for the computer to distinguish actions of a user intended as digital ink from actions intended to invoke operations of or provide commands to the computer system. It is desirable that a user interface for a pen-based computer system be able to interpret movement of the stylus in a consistent, unambiguous fashion, so that users can enter data or invoke commands quickly and without frustration. It is also important that the user interface be intuitive and easy to remember, so that the user can readily determine how to tell the computer that he or she intends to invoke a command, or make digital ink (e.g. enter text via handwriting)

One example of a known pen-based computer system is the Newton™ MessagePad™ manufactured by Apple Computer, Inc., Cupertino, Calif. This device is a small, hand-held device with a liquid crystal display (LCD) screen, no keyboards or buttons (other than an on-off switch), and a stylus that is used to interact with the system. The user manipulates the stylus to enter handwriting, tap on regions of the screen, or draw pictures. The Apple Newton attempts to interpret patterns of markings by the stylus on the screen or "gestures" as commands. A gesture is a symbol or pattern of markings that the user can draw with a stylus (or finger) that is recognized by the system as a command, rather than as text or a picture.

For example, in the Apple Newton a squiggly gesture made on top of a word like this: word means "delete this" and (if interpreted properly) causes the deletion of the object or text upon which the squiggly mark was drawn. However, the computer might interpret the marking as the character "W", and the result would be a misspelled word "wordW". Likewise, a caret gesture like this: to‸close means to "insert space". Moreover, the user must remember that a squiggly gesture means "delete", and that a caret gesture means "insert", since there is no readily apparent means for telling the user what the gestures are if forgotten.

Stated in other words, gestures are "hidden" from the user and can be ambiguous. That is, there is no way for a user to determine what particular gesture to draw when a particular command is desired, except to be told or remembered. Once learned, however, gestures can be easily forgotten, if not repeatedly used. Another disadvantage is that of recognition. Gestures can be and frequently are misrecognized. The results of a misrecognized gesture range from simple annoyance to unintended loss of data.

Even with traditional keyboard and/or mouse-type graphical user interfaces, there is sometimes ambiguity as to the function of an icon or the manner for invoking a command for a data object displayed on screen. For example, in a mouse-driven GUI, if users wish to delete a word from within a sentence, they must remember that they must (1) first select the word to be deleted by pointing and dragging the cursor (thereby forming a "selected data object") and (2) then invoke the appropriate "delete" command by either (a) remembering the keyboard equivalent command, (b) finding the command from a drop-down or pop-up menu which may be remotely located on the screen relative to the selected data object, or (c) activate a disassociated "delete" icon somewhere on the screen, assuming they can find it.

While all of such methods (keyboard command, menu command, icon command) are presently employed with success in modern application computer programs, it would advance the art of GUI design if the object itself upon which an action is desired could provide the user with the options to invoke actions in a concise, consistent and uniform methodology. Such an advance would not force users to remember keyboard commands, locate commands in menu located away from the object, or locate appropriate action icons located remotely from the object. In yet other words, it would be desirable that selected data objects themselves provide a context for prompting and receiving user interaction.

In order to reduce ambiguity to some degree, some application programs for the Apple Newton Message Pad system employ fixed-position icons that display pop-up menus of selectable items when tapped. (A "tap" is a form of gesture wherein the user lightly strikes a spot on the display screen with the stylus.) These fixed-position icons are typically a small, diamond shaped region (e.g. ♦) whose very shape signifies that it is a "hot spot". It has proven easy for users to remember that the diamond icon means "tap here for display of selectable options."

However, such pop-up menus are associated with a series of selectable items, and are not context-sensitive. There is no predictable one-to-one connection between an object and an icon. In other words, selected data objects themselves do not provide a context for prompting and receiving user interaction. Moreover, such pop-up menu icons do not completely duplicate gesture functionality, and cannot be moved or repositioned.

It is expected that the use of pen-based computing systems will increase in the foreseeable future because of the flexibility afforded by such systems to people who either cannot or do not wish to type using a keyboard. Moreover, pen-based computers offer the advantage of greater portability since no keyboards or pointing devices are required other than a simple stylus. It is foreseen that even future desktop computer systems may utilize flat panel display screens that serve both as an upright display when a user wishes to use a keyboard and/or mouse for interaction, but may also be removed from a stand, placed in a lap or on a desktop, and employed like "digital paper" with a stylus, as the user enters "digital ink". Therefore, there is a need for computer operating systems that offer increased command accessibility and reduced ambiguity, especially for pen-based computer systems, but for that matter with any types of computer systems that rely upon undiscoverable or easily forgettable actions that are associated with particular icons.

SUMMARY OF THE INVENTION

The present invention overcomes the foregoing described difficulties with present computer user interface design by providing action handles or icons in association with data objects, typically displayed in close proximity to a related data object displayed on the screen. The action handles provide a method for performing actions on text and other data objects with a pen or other pointing device, and also a method for displaying a context menu for the text or other objects. A context menu, for purposes of the present application, is a menu of commands that apply to only a particular object within a particular context, and that pops up on screen over, or in proximity to the particular object.

Briefly described, the present invention comprises computer systems and methods that comprise displaying a small icon or "action handle" next to or in close proximity to a data object, such as a selected text item, or a calendar item, or a list of other data items, or at an insertion point within a range of text. The action handles receive event messages indicative of a tapping or dragging action with a stylus or other pointing device such as a mouse. In response to a tapping action, preferably a context menu displaying commands available for the object or text is displayed. In further response to selection of a command in the context menu, the system carries out the appropriate selected command. In response to a dragging action, preferably the selected object is moved to a new location.

For example, an action handle next to an address in an address book can be dragged to copy or move the address to another application, for example, to an electronic mail ("e-mail") application to send mail to a person. Or, the action handle can be tapped to cause display of the context menu that offers commands that are appropriate in a given context, such as "delete", "print", or "edit", that act on the address.

As another example, the action handle associated with a selected text data object might be used to move the text around by dragging, and might contain commands in its context menu such as "delete", "make upper case", "make bold face". Similarly, an action handle associated with an insertion point (which may include a draggable cursor) can be dragged to select text, which then yields an action handle associated with the selected text. The selected text action handle itself, like the draggable cursor action handle, can be tapped to reveal commands such as "insert space" or "start new line".

Preferably, the action handle will contain graphical information to assist in identifying the object in question. For example, the action handle for an address item in an address book might look like a business card.

More particularly described, the present invention relates to methods and systems for displaying and controlling actions relating to a data object displayed on a display in a computer controlled display system having a display coupled to a central processing unit (CPU). The methods and systems include the steps of generating a graphic image associated with the data object on the display, and generating a graphic image of an action handle proximate to the data object. An activatable region on the display associated with the action handle graphic image is provided. A user positions a pointer on the display using a pointing control device operatively associated with the CPU, which thereby provides a signal to the CPU upon correspondence of the pointer with the activatable region. In response to a signal indicative of a first type of interaction between the pointer and the activatable region, the method involves displaying at least one selectable command relating to the data object. In response to signal indicative of a second type of interaction between the pointer and the activatable region, the method involves moving the graphic image of the data object on the display.

In the preferred embodiment, the first type of interaction is a tap, while the second type of interaction is a drag. These types of interactions are particularly suitable for pen-based computer systems.

The preferred step of displaying at least one selectable command comprises displaying a context menu on the display including a plurality of selectable commands. In the context menu, there is provided an activatable second region on the display associated with the selectable command. In response to interaction between the second region and the pointer, a computer command corresponding to the selectable command is executed. Preferably, again, the interaction between the pointer and the second region comprises a tap.

Still more particularly described, the step of moving the graphic image of the data object on the display comprises moving the graphic image of the action handle on the display during a dragging operation from a first position to a second position on the display. In response to a release command provided at the second position on the display (typically by withdrawing the pen or stylus from the display screen), a predetermined operation is carried out, e.g. the graphic image of the data object is moved to a new position on the display associated with the second position, or if the object is "dropped" onto a clip box region, the data object is copied.

In the invention, the data object may be a data item associated with a predetermined application program. For example, the data item could be a selected string of text associated with a text processing application program, or a calendar entry associated with a calendar application program.

It will be appreciated that the invention has utility for both pen-based and mouse-based (e.g. desktop) computer systems. Advantageously, action handles are very discoverable and cannot be forgotten since they are always displayed in association with the data object. To find out what command are available in the context of a particular object, the user can simply tap on the action handle to display the context menu, and then selects an appropriate command from the list of choices in the context menu. Execution of a command in the context menu operates like other menus.

Action handles also have no ambiguity about them. The actions available at any given time are always displayed by tapping the action handle to reveal the context menu, and if the item is movable, the action handle icon, as it is dragged with the stylus or pointing device on the screen, can change its graphical properties to indicate that movement is permitted. If movement is not permitted, the action handle simply never moves or pops back to the original location, indicating non-movability to a particular location. This would be handy in applications wherein a particular data item cannot be discarded, as when a user attempts to drag the non-discardable data object to the trash can or a delete icon, and the icon pops back to its location original.

Action handles therefore eliminate the need for gestures. Action handles always provide a visible place where the stylus always acts as a pointing device rather than as a pen (that is, no digital ink is created when touching an action handle). In text entry areas, mouse operations such as drag-and-drop and selection are very difficult for a computer to distinguish from inking and writing operations. Action handles solve this problem.

It will thus be appreciated that action handles work with any type of pointing device, and provide a mechanism whereby a user can readily determine that an action is possible. This solves the problem of creating context menus in computer application programs that rely upon a "right mouse" button (for which fingers and pens have no analog) for activation of a context menu. With an action handle, there is always a visible indication of what a user can click or tap on to get a context menu.

The action handle upon its creation can indicate a visible encapsulation of an object being manipulated. In other words, the icon can represent the object for actions to be taken with the object. Action handles further permit a single-tap or click method to cause display of the context menu. This contrasts with certain double-click or other multiple tap or click methodologies which are difficult for users to remember.

The action handle provides a single focus of the user's attention for all actions on an object, both dragging and command execution. Again, this helps eliminate ambiguity and promotes consistency in the user interface across different applications and through the operating system.

Action handles, as contemplated by the present invention, are automatically generated and exposed in the appropriate context. If any particular action is available for an particular object, the action handle is displayed. Thus, the user need not remember any gestures, and can always find out what action to take by either dragging the action handle or tapping the action handle. Even common insertion-point icons or cursors can automatically appear in edit controls.

Accordingly, it is an object of the present invention to provide an improved graphical user interface for computer systems.

It is another object of the present invention to provide an improved graphical user interface particularly suitable for pen-based computer systems.

It is another object of the present invention to provide an improved graphic user interface for pen-based computer systems that reduces ambiguity in the possible actions that can be taken with respect to an object displayed on a display screen.

It is another object of the present invention to provide an unambiguous method for selecting text and other objects in a pen-based computer system graphic user interface.

It is another object of the present invention to provide for an improved user interface for pen-based computers that allows a computer system to distinguish actions of a user intended as digital ink from actions intended to invoke operations of or provide commands to the computer system.

It is another object of the present invention to provide a pen-based user interface for a computer system that is able to interpret movement of a pen or stylus in a consistent, unambiguous fashion, so that users can enter data or invoke commands quickly and without frustration.

These and other objects, features, and advantages of the present invention may be more clearly understood and appreciated from a review of the following detailed description of the disclosed embodiments and by reference to the appended drawings and claims.

BRIEF DESCRIPTION OF THE DRAWING FIGURES

FIG. 1 illustrates the general architecture of a handheld pen-based computer that is constructed in accordance with and utilizes the teachings of the present invention to provide action handles in association with data objects.

FIG. 2 illustrates an exemplary display screen for the computer of FIG. 1, that illustrates various relationships of action handles and associated data objects.

FIG. 3 illustrates exemplary static action handles utilized in the preferred embodiment of the invention to indicate different types of data objects or items.

FIG. 4 illustrates exemplary dynamic (in dragging motion) and status indicating action handles or "drag icons" utilized in the preferred embodiment of the invention to indicate status of a drag or a drag-and-drop operation.

FIG. 5 illustrates a general drag-and-drop operation utilizing an action handle.

FIG. 6 illustrates a drag-and-drop operation to a clip box, for copying a data object associated with an action handle.

FIGS. 7A and 7B illustrates pop-up or context menus that are generated in the preferred embodiment of the present invention when a user taps on an action handle associated with a predetermined application and a predetermined data item, respectively.

FIGS. 8A–8C illustrates a drag-and-drop operation applied to an action handle associated with a range of selected text in an exemplary text-based application program, in accordance with the invention.

FIG. 9 illustrates an action handle that is "pinned" to a border of a window in accordance with the invention, indicating that the associated data object has been scrolled off-screen.

FIG. 10 illustrates an action handle associated with a small range of selected text of a size approximating that of the action handle itself, displayed beneath the selected text, in accordance with the invention.

FIGS. 11A–11G illustrates an action handle in association with a flashing cursor data object, for purposes of text selection and manipulation, in accordance with the invention.

FIG. 12 is a table illustrating a general class of objects (a rectangle) with associated instance variables, messages, and methods, in connection with object-oriented programming.

FIG. 13 is a table illustrating an exemplary Icon class of objects with associated instance variables, messages, and methods, with corresponding subclasses of Program Icon and Action Handle Icon, in accordance with the invention.

FIG. 14 is a table illustrating a hierarchy of object classes utilized in constructing a computer program that implements the preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring now to the drawings, in which like numerals illustrate like elements throughout the several views, FIG. 1 illustrates a pen-based computer system 10 for generating graphic images on a graphic display 18 and receiving user interactions by way of a pen or stylus 12. Although the present invention is described in conjunction with a pen-based computer, it will be appreciated that the present invention may be utilized in conventional (desk top or portable) computer systems that use a keyboard or cursor keys to position a cursor on the display screen, as well as a mouse, trackball, or other pointing device that moves a cursor on the display and often includes a switchbutton to indicate a command.

In the following detailed description, numerous details are provided such as computer display system elements, object definitions, display formats, sample data, etc. in order to provide an understanding of the invention. However, those skilled in the art will understand that the present invention may be practiced without the specific details. Well-known circuits, programming methodologies, and structures are utilized in the present invention but are not described in detail in order not to obscure the present invention.

Certain of the detailed descriptions which follow are presented in terms of exemplary display images, algorithms, and symbolic representations of operations of data bits within the computer's memory. As will be known to those skilled in the programming arts, and particularly those skilled in the object-oriented programming methodologies (about which more is explained later), these algorithmic descriptions, class descriptions, messages, notification descriptions, and graphic exemplary displays are the means used by those skilled in the art of computer programming and computer construction to convey teachings and discoveries to others skilled in the art.

For purposes of this discussion, an "algorithm" is generally a sequence of computer-executed steps leading to a desired result. These steps require physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical, magnetic, or optical signals that are capable of being stored, transferred, combined, compared, or otherwise manipulated. It is conventional for those skilled in the art to refer to these signals as bits, values, elements, symbols, characters, images, terms, numbers, or the like. It should be kept in mind, however, that these and similar terms should be associated with appropriate physical quantities inside the computer and that these are merely convenient labels applied to these physical quantities that exist within the computer.

It should also be understood that manipulations within the computer are often referred to in terms such as adding, comparing, moving, etc. which are often associated with mental operations performed by a human operator. It must be understood that no involvement of a human operator is necessary or even desirable in the present invention, since the operations described herein are machine operations performed in conjunction with a human operator or user that interacts with the computer. The machines used for performing the operation of the present invention, as will be understood, include general purpose digital computers or other similar computing devices.

Furthermore, it should be kept in mind that there is a distinction between the methods, steps, or operations carried out by a computer, and the method of computation itself. The present invention does not involve a method of computation. The present invention rather relates to methods, steps, or operations for a computer and processing electrical or other physical signals to generate desired physical signals and display results and interactions. As illustrated in FIG. 1, the present invention also relates to apparatus 10 for performing these operations. This apparatus may be especially constructed for the described purposes, e.g., a touch-sensitive pen-based computer system, or it may comprise a general purpose computer that is activated or operated by computer programs stored in the computer.

Furthermore, it should be understood that the programs, algorithms, objects, etc. described herein are not related or limited to any particular computer or apparatus. Rather, various types of general purpose machines may be used with programs constructed in accordance with the teachings herein. Similarly, it may prove advantageous to construct specialized apparatus to perform the method steps described herein by way of dedicated computer systems with hardwired logic or programs stored in nonvolatile memory such as read only memory.

OBJECT ORIENTED PROGRAMMING

Those skilled in the art will understand and appreciate that the best mode for implementing the present invention is via object-oriented programming languages and techniques. Object-oriented programming techniques are increasingly utilized by those skilled in the art of computer programming because they provide a powerful way to view the interaction of a user with a computer system. Since people interact with objects, it is logical to attempt to write computer program so that the "objects" with which a person interacts on a computer screen have counterparts "inside the computer", so to speak.

For example, one particular "object" that people often work with on computer screens is called a "window". A "window" is a rectangular area on a display screen that is used to display information or receive interactions. Indeed, the display screen itself is a window of sorts into the computer. A window receives user input from a keyboard, mouse, or pen, and displays graphical output on its surface. A computer program is responsible for generating the window, and responding to user interactions. A window often contains the computer program's title bar across the top of the window, menus, sizing borders, and perhaps other objects.

The surface of a window may contain additional, smaller windows called "child windows", which are subspecies or subclasses of windows. Child windows may take the form of push buttons, radio buttons, check boxes, text entry fields, list boxes, and scroll bars, and the like. These objects do not truly exist as physical objects, but when displayed on the computer screen may be pressed or clicked or scrolled like they have physical existence. A user sees a window as an object on the screen, and interacts directly with the object by pushing "a button", or scrolling a scroll bar, or tapping a handle.

A skilled object-oriented programmer's perspective is analogous to a user's perspective. A window generated by a computer program receives user input in the form of messages to the window. The messages are generated by an event handling routine that is responsive to user input such as tapping, clicking or dragging a cursor with a stylus or mouse.

The following is a brief example of an object-oriented programming methodology for resizing a window utilized in the Microsoft WINDOWS™ graphical user interface operating system. Consider a word processor application computer program that displays text in a window. The window may include a "size box", which is a control located at the lower right-hand corner of the window. If selected and dragged, the size box causes the size of the window on the display screen to change to larger or smaller. If the program's window is resized, the word processing program application program will reformat and move the text on the display screen to fit within the window when the window is resized.

The computer's operating system program generally handles the details of resizing the window, and the application program running under control of the operating system program responds to the system function of resizing the window. The word processing program "knows" when its window is resized because a message is passed to it indicating a resizing operation, and the application program responds by repositioning the text within the available window. In the WINDOWS™ operating system, when a user resizes a window, the operating system sends a message to the application program indicating the new window size. The program then adjusts the contents of its window to reflect the new size.

Stated in other words, the operating system program sends a "message" to the application program by calling a function (a procedure or a method) within the program. The parameters of this message describe the particular function to be executed by the computer.

The idea of passing a message to a program is tantamount to making a call to a function within the program. Such a function call can be made to the operating system, or to an application program. Those skilled in the art will understand that such mechanisms are how a program opens a disk file, for example, or scrolls text from an area that is "off screen" into view within the window. An operating system may make calls to a program by passing it messages, in a similar manner that the application program makes a call to the operating system.

Every window object that a program creates has an associated window procedure (also called a "method"). This window procedure or method is a function that could be either in the program itself or in a linked library of appropriate routines that can be executed upon a call. The WINDOWS™ operating system sends a message to a window by calling a window procedure. The window procedure does some processing based on the message passed to it, and then returns control to the operating system, possibly with a "notification" or other response. (A "notification" is merely a message passed back to a calling program, and usually indicates status information or the results of an operation.)

A window displayed on a display screen is based on what is known as a "window class" of objects. The window class of objects identifies a particular procedure that processes messages passed to that window. The use of a window class allows multiple windows to be based on the same window class, and thus use the same window procedure. For example, all control buttons in programs that utilize the WINDOWS™ operating system are based on the same window class. A window class is associated with a window procedures that processes messages to all button windows.

In object-oriented programming, an "object" is a combination of computer program code and data. A window is a type of object. The computer program code carries out tasks associated with the object. The data is the information retained by the window procedure and information retained by the WINDOWS™ operating system, for each window and window class that exists.

A window procedure processes messages to the window. Very often these messages inform a window of a user input from a keyboard, mouse, or pen. This is how a child window that is a button "knows" that it is being "pressed", or an icon "knows" that it is being dragged. Other messages tell a window when it is being resized or when the surface of a window needs to be repainted.

When a program begins execution in such an operating system as the WINDOWS™ operating system, the operating system creates a "message queue" for the program. This message queue stores messages to all the various windows a program may create. The program includes a portion of program code called the "message loop" to retrieve the messages from the queue and dispatch them to the appropriate window procedure. Other messages may be sent directly to a window procedure without being placed in the message queue.

Preferred Handheld Computer System

With the preceding background in mind, turn next to FIG. 1 for a discussion of the preferred embodiment of the handheld pen-based computer **10** constructed to carry out the methods of the present invention. The pen-based computer **10** comprises various components, which are illustrated graphically for purposes of illustration. First, a rectangular liquid crystal display (LCD) **18** occupies the front surface of a housing **19**, and provides a surface upon which graphic images are displayed. Preferably, the LCD display **18** is touch or electromagnetically sensitive. A stylus **12** that is held by a user is used to write, draw, or gesture upon the surface of the display **18** to allow user interaction with the computer.

The computer **10** is preferably constructed around a central processing unit (CPU) **20**, a memory **21**, and an input/output (I/O) circuit **22**, all of which are physically contained within the housing **19** but are shown externally in FIG. 1 for ease of understanding. It will be understood that the I/O circuit **22**, CPU **20**, and memory **21** are those typically found in most general purpose computers. Indeed, computer **10** is intended to be representative of a broad category of data processing devices. Thus, although there is no keyboard or other pointing device shown in FIG. 1, it will be understood that such types of computer systems are also suitable for use with the present invention.

The I/O circuit **22** is used to communicate information in appropriately structured form to and from other portions of the computer **10**. For example, the I/O circuit **22** drives an optical port **25** which is employed to communicate information optically to similar optical devices. The I/O circuit further connects to a serial port **27**, which allows electrical connection of peripheral devices such as communication lines to a desktop personal computer system, modem, or other device. The I/O circuit further connects to a PCMCIA slot **29** which allows usage of a PCMCIA memory card or other peripheral PCMCIA device such as shown at **31**.

The display **18** is shown coupled to the I/O circuit **22** and is used to display images, data, etc. generated by the CPU **20** in accordance with the present invention. The display, as will be known to those skilled in the art, is a rectangular array of picture elements or "pixels", beginning at an origin (x=0, y=0), and extending across the width of the display screen to the rightmost extent (x=W), and downwardly to the lowermost extent (y=H) of the display screen.

In accordance with the invention, shown on the display screen **18** are two particular graphic display elements or items that in general characterize the present invention. Firstly, there is shown an action handle or icon **40** positioned adjacent to a text object **42**. Preferably, both of these graphic display elements are objects in the object-oriented computer programming sense.

In accordance with the present invention, an action handle or icon **40** is displayed in proximity to (such as adjacent or atop) an associated data item or object such as the text **42**; user interaction with the action handle **40**, and therefore with the associated data object **42**, is permitted in various ways described herein, without requiring gestures. Data items that utilize the present invention will always include an associated action handle in proximity on the display screen.

In the example shown in FIG. 1, exemplary text object **42** happens to be a string of text that is displayed in a rectangular region on the display screen **18** defined by the dotted box. (It will be understood that the dotted box is not displayed on the display screen of the computer, but merely signifies an exemplary data object—the text contained within the rectangular boundaries shown—for purposes of ease of understanding. Moreover, the invention contemplates various types of data objects, not merely text objects, e.g. a cursor object, a drawing object, an icon object, etc.)

The action handle **40** typically is relatively small compared to the data item, although size is not critical and no limitation as to size is intended. The size of the action handle **40** should preferably be such that a user can readily locate the action handle associated with any data object on the display screen. Indeed, the presence of an action handle indicates the presence of a selectable data item or object and the ability for the user to interact with the data object.

The exemplary action handle **40** shown in FIG. 1 comprises an activatable object, region, or "hot spot" on the display screen. This hot spot is bounded on the display screen by a predetermined region along the x-axis between x=HS1 and x=HS2, and along the y-axis between y=HS3 and y=HS4. If the user places the tip **13** stylus **12** within the hot spot region defined by these coordinates, the touch-sensitive circuitry of the display screen **18** will detect the presence of the stylus within the hot spot and generate an appropriate message. In a pen-based computer system, two particular actions of the stylus relative to the screen are contemplated—a tap and a drag. A tap occurs when a user quickly places the tip of the stylus within the hot spot and lifts it up, tapping the surface lightly on the display screen. A drag occurs when the user places the tip of the stylus into the hot spot, and rather than lifting the stylus, leaves the tip of the stylus in contact with the surface of the display screen and moves the stylus along a path on surface of the screen. In accordance with the present invention, the computer will generate appropriate graphical symbols on the display screen that will follow the tip of the stylus around on the display screen as if the user were "dragging" the action handle.

The foregoing actions of tapping and dragging allow the computer **10** to differentiate between actions that are intended by a user as an interaction with the data object or with the computer system, versus actions that are intended to constitute the entry of data such as handwriting or drawing on the display screen, both of which generate "digital ink" such as shown at **14**. As has been previously described, "digital ink" **14** is an illuminated region of pixels on the display screen generated by the computer system in response to the movement of the stylus tip along the touch-sensitive surface of the screen.

Before turning to another subject, it will be understood that in accordance with the invention, an object such as the exemplary text object **42**, and its associated action handle such as the action handle **40**, are selectably repositionable on the screen. In other words, should the object be moved to another location on the screen, as by invocation of a command, a scroll of the window, a dragging operation, etc., the action handle maintains its relative position in accordance with the associated data object. No matter where a particular data object appears, if it has an action handle associated therewith, the action handle will be displayed in appropriated proximate juxtaposition. Those skilled in the art will understand that programming for the computer to implement the present invention will need to take into account the relocatability of a data object, and provide for corresponding relocation of the associated action handle.

FIG. 2 illustrates various different relationships of action handles **40** with associated data objects **42**, and the preferred relationship contemplated as the best mode by the present inventors. Several different embodiments of action handles and associated data objects are shown. The upper region of the display screen **18** includes a list of data items or objects, which in this case happens to be a list of names and telephone numbers in an Addresses window. An exemplary name in the list, "Bill Clinton", is shown highlighted as a selected data object **42***a*, with an associated action handle **40***a*. In this example, the action handle **40***a* is a small, "Rolodex" card-shaped icon positioned immediately to the left of the associated data item **42***a*.

In the central region of the display screen **18** in FIG. **2**, there is shown a selected group **42***b* of two names, "Berman, Eric" and "Bishop, Byron". Both names are highlighted, indicating that the data items are selected. The associated action handle **40***b* comprises an icon bearing the resemblance of a stack of multiple sheets of paper. In the preferred embodiment, this particular icon indicates selection of multiple items. Thus, it will be understood that a data object can comprise plural data items considered as a group; an action handle such as **40***b* associated with such a group operates upon and with the group of data items.

In the lower portion of the display screen **18** in FIG. **2**, there is shown a data object **42***c* comprising string of highlighted text, which in this case consists of the words "WinPad provides drag icons by which the user can manipulate selections and objects on the WinPad screen". The associated action handle **40***c* consists of a small rectangular box containing the lowercase letter "a" and is positioned atop the selected text.

In the upper right hand of FIG. **2**, there is shown a data object **42***d*, which in this case comprises a "clip box" for carrying out clipboard operations such as copying. An associated action handle **40***d* consists of a small rectangular box positioned inside the clip box. As contemplated by the invention, if the user wishes to copy certain information to a different page or application, the user can use the clip box as a temporary holding area for the information. For example, the user can copy a name (such as the name "Bill Clinton") to the clip box by dragging an action handle **40***a* associated with the name to the clip box. The data (the name) will be copied into a memory buffer, and available for "pasting" into another program or into another object. The presence of an action handle such as **40***d* shown in the clip box indicates that a data item has previously been "copied" into the clip box. Preferably, the action handle can be dragged out of the clip box to perform a paste operation.

It will therefore be appreciated that the present invention contemplates provision of various types of action handles, in association with various different types of data objects. In the preferred embodiment of the present invention, action handles **40** are preferably positioned on the display **18** as follows:

For a single data object, such a single name **42***a* in the list, the action handle is preferably placed to the left of the object. (Note—preferably, this icon is allowed to scroll off screen if the associated object is scrolled off screen.)

For a group of objects, such as a selected series of addresses **42***b* in the address list, the action handle is placed to the left of the objects, but vertically centered through the vertical extent of the selected group of objects. (Note—preferably, this icon is allowed to scroll off screen only if all associated objects are scrolled off screen.)

For an insertion point, which is typically a flashing caret positioned somewhere in a string of text or on a drawing figure, the action handle is placed just beneath the baseline of the insertion point cursor or beneath the baseline of the text.

For a selected range of text on a single line, the action handle is preferably placed immediately above or below the text, wherever there is more visible room, centered horizontally with respect to the selected text, so as to avoid obscuring the text.

For a selected string of text that spans more than one line, such as shown at **42***c* in FIG. **2**, the action handle is preferably placed in the center of a bounding box of the selected text. A bounding box is the smallest rectangle that would enclose the selected text.

With the exceptions noted above, if any of the foregoing rules for action handle placement cause the action handle to be off screen or outside the visible bounds of the window in which the object or text is displayed, the action handle is preferably moved vertically and/or horizontally until it is just inside a bounding rectangle of the window or screen on the display. If only one coordinate (x or y) exceeds the bounds of the window, then only that one coordinate needs to be adjusted.

The final rule is most likely to be utilized when the window constitutes a scrollable pane, which can occur if a data object is selected by the user, and the pane is scrolled so that the selected text and its associated action handle are scrolled off screen. Preferably, then, the action handle is displayed in the manner described even though the associated data object is not visible to the user. This is described in greater detail below.

FIGS. **3** and **4** illustrate exemplary action handles **40** in preferred static and dynamic forms, respectively. In FIG. **3**, there is shown a series of different exemplary action handles **40***a* through **40***h* whose icons indicate the nature of the associated data object, which are listed. Similarly, FIG. **4** illustrates the display of dynamic, status-indicating icons that are displayed as an action handle is dragged from one location to another on the display screen. Note that each different action handle has a different graphic symbol or icon associated therewith. The graphic nature of the icon indicates the nature of the associated item.

In FIG. **3**, an action handle icon **40***a* comprising a lowercase letter "a" is utilized to indicate a text data item. The icon **40***b* is preferably a small list to indicate a to-do item. An envelope icon **40***c* indicates a message. A "rolodex card" icon **40***d* is used to indicate an address. A miniature calendar icon **40***e* is used to indicate a calendar item. A pencil icon **40***f* is used to indicate a note data item. A stacked sheets of paper icon **40***g* is used to indicate that multiple objects have been selected. An insertion point/caret icon **40***h* is preferably used in association with an insertion point (flashing bar) to indicate actions that can be taken with respect to text (e.g., dragging to select text, or tapping to invoke predetermined commands).

The action handles shown in FIG. **3** generally are displayed statically in the present invention in association with a data item of a corresponding type. The action handle remains displayed as long as the data object is visible on the screen in the form shown.

Action Handle Drag and Drop Operation

It will be recalled that one of the allowed actions is to drag an action handle, for purposes of moving the associated data item, or copying it, or invoking another action. As will be known to those skilled in the art, it is a customary practice in many computer graphical user interfaces for a program to generate and display a different icon from the static icon when an object is dragged with a pointing device such as stylus or mouse. The use of a different icon during motion generally provides certain predetermined status information about the action undertaken, for example that the item associated with the icon has been selected and capable of being moved, or that a time delay is occurring, or that a given action is not permitted. For purposes of the present invention, an icon displayed during an action such as dragging is called a "dynamic" or "status indicating" icon, and may also be called a "drag icon."

Drag icons are preferably utilized in the present invention during certain predetermined operations, namely, when an action handle is dragged or otherwise activated. FIG. **4** illustrates various dynamic action handles or drag icons **40***a*–**40***g* employed in the preferred embodiment. A selected one of these drag icons is generated and displayed "underneath" the stylus point or other pointer as the stylus or pointer is (1) first used to point to and select an action handle by placing the stylus tip onto the hot spot of the action handle (or placing a pointer onto the hot spot and clicking to select the action handle, if a mouse or trackball is used), and (2) then moved about on the display screen to "drag" the action handle. Generally during a drag operation, the static action handle (as shown in FIG. **3**) remains displayed in position with its associated data item, and the computer CPU **20** generates one of the exemplary drag icons as shown in FIG. **4** until the user "releases" the drag icon by withdrawing the stylus away from the display screen (or by clicking or other appropriate action if another type of pointing device is used).

The exemplary drag icons of FIG. **4** provide visual feedback to a user about the status of the current operation, such as a drag and drop operation. For example, a copy pointer drag icon **40***a* indicates that a copy operation has been invoked, that the selected data item has been copied, and that a copy of the selected item is being moved. A multiple objects pointer drag icon **40***b* indicates that a selected group of objects is being moved from one location to another.

The user can also determine that the data item can not be dropped at the current pointer location (e.g. when the user attempts to drop the selected data item on or in an incompatible or impermissible location) when the drag icon appears as a circle with a slash through it, as shown at **40***c*.

Other types of move and copy type drag icons are shown at **40***d*–**40***g*. In the preferred embodiment, a user can determine whether a copying operation or moving operation has been activated by the presence or absence of a plus sign (+) associated with the drag icon. For example, the "copy" type drag icons **40***d*, **40***e* include a plus sign (+), while similar but "move" type drag icons **40***f*, **40***g* lack the plus (+) sign.

Preferably, the selection of a copy type icon or a move type icon is determined at the time an object is dropped, and depends upon the drop target. For example, if a drag icon is moved onto a storage directory or subdirectory, the plus sign (+) may be generated to indicate that a drop of the icon would result in the item being copied into that directory or subdirectory.

Refer now to FIG. **5** for a discussion of the manner in which a user of a pen-based computer interacts with an action handle for a dragging operation, and how the static action handles and drag icons are generated. In the figure, the action handle **40** is being dragged with a stylus **12**, from its initial position to a rest position at **53**. In this example, the data object is a calendar item **42** indicating "12:00—Project status meeting with Staci", and has a corresponding (static) action handle comprising a miniature calendar. The static action handle **40** remains in position during the operation, displayed in proximity to the data object. As the stylus **12** is moved to drag the action handle, a drag icon **40'** is generated at successive positions along the trajectory **48** of the stylus, beginning immediately after the tip of the stylus leaves the area of the static drag icon at **51** until the resting position at **53**, which is at the calendar time 5:00 PM. At the resting position **53**, assume that a location where the selected data item may properly be deposited or "dropped", e.g. at the time 5:00 PM, which occurs upon release or withdrawal of the stylus (or clicking or tapping, if desired). If the end result of the operation is a proper move operation, upon release the static action handle **40** and its associated data item **42** appear at the new location, and are deleted from the old, initial location.

It will now be appreciated that, in the preferred embodiment, a user can use action handles to copy or move information anywhere within a supported series of applications operating within the computer system. The approach utilized in the present invention is considered a "drag-and-drop" approach, which is generally known to those skilled in the art. The drag-and-drop approach to data movement and manipulation is very useful when a user needs to change or duplicate information quickly. In the calendar example of FIG. **5**, if a user wishes to change a meeting to a different time of day, he or she can reschedule it by dragging the action handle **40** associated with an appointment time within a calendar to a new time, or a new date. During the drag operation, an appropriate drag icon is generated and displayed.

Those skilled in the art will understand that a drag-and-drop operation depends upon the particular application and context. For example, in a calendar application program, information that is dragged within a page is considered moved, whereas in a different application, such as a "to-do" list, the information is copied rather than moved. Preferably, therefore, the system is programmed to presume that a move is intended if the drag-and-drop operation occurs within a given application, but that a copy operation is intended if the data item is posted to a clip box to be copied, or moved to a different application program to which the data can properly be copied.

Refer now to FIG. **6** for a discussion of the use of action handles to move a data item from one application to another. As previously described, the present invention supports a copying operation using the action handle. The clip box **45** provides a region within the display **18** for providing a "clipboard" operation. If the user wishes to copy the information associated with an action handle to a different page or application, the user can use the clip box as a temporary holding area for the information.

For example, assume in FIG. **6** that the user wishes copy a data item comprising a note **42** "Product Proposal" from a Notes application to a Mail application. The user first drags the action handle **40** associated with the data item along the path **58** to the clip box **45**, activates the Mail application, and then drags from the clip box into the Mail application. A clip box action handle **40***d* appears in the clip box **45** after this first drag operation to indicate that data has been copied into the clip box. The movement into the Mail application is accomplished by dragging the action handle **40***d* out of the clip box along a path **59** to a new location.

In the preferred embodiment, the clip box **45** only holds one item of information at a time. If a second item is copied to the clip box by dragging and dropping, then the existing information is deleted. However, information can be copied from the clip box multiple times, as the information is not deleted until another item is copied on top of it.

Thus, to copy or move information between pages or applications, the user:

Drags the action handle of the associated data object to the clip box **45**.

If it is desired to copy or move information to another application, then switch to the other application by using the stylus, pointing device or other command to activate the application.

Activates other controls in the computer system to display the dialog, or page, where the information is to appear.

Drag the action handle from the clip box **45** to its new location.

If the user is moving information, the user may switch back to the originating or first application and delete the original information.

Action Handle Tapping or Other Selection

As previously described, a particular action contemplated in the present invention is that of tapping an action handle with a stylus. The preceding diagrams have illustrated operations involving dragging an action handle. In the preferred embodiment of the present invention, tapping an action handle causes display of a "pop-up" menu, also called a "context" menu, that display various types of actions that may be selected with respect to the particular data object associated with the action handle. Those skilled in the art will understand that in computer systems that utilize pointing devices such as a mouse or trackball, the same considerations will apply, except that the operation is that of pointing and clicking as opposed to tapping.

FIG. **7** illustrates two exemplary types of pop up menus contemplated in for use in the present invention. FIG. **7**A illustrates an application-oriented pop-up or context menu **70***a* that appears when application action handle (e.g. a Notes action handle **40**) is tapped. For purposes of the present discussion, an "application" action handle is an action handle that is positioned in proximity to a data item corresponding to an application computer program, such as a word processor, a note taking utility, a calendar program, an address list manager, etc. In accordance with the invention, the application pop-up menu **70***a* includes, in a predetermined hierarchy, a series of selectable actions that may be activated by the user. In FIG. **7**A, the selectable actions include Properties . . . , Delete, Clear Ink, Send . . . , and Print . . . Preferably, the pop-up menu appears on the display screen in response to a tap of the action handle, overlaid upon other information, and remains until (a) the user taps somewhere on the screen outside the menu or presses the escape key, or (b) one of the selectable activities has been activated by a tap upon the appropriate desired action. For example, to initiate a printing operation, the user would tap the Print . . . region, which would be highlighted, flashed, and a printing operation would commence.

Other types of pop-up menus are contemplated. In FIG. **7**B, a data item pop-up menu **70***b* is illustrated. A data item pop-up menu is displayed when an action handle associated with a data item is tapped. In the example shown, there is a list of names, including the name "Benson" shown in highlighting (the other names are obscured by the pop-up menu **70***b*, although their action handles are visible). The data item pop-up menu also contains, in a predetermined hierarchy, a series of selectable actions that may be activated by the user.

The pop-up menu **70***b* displays various actions that are available for this particular data object. For example, in FIG. **7**B, the operations of Edit . . . , Delete, Send . . . , Print . . . and Help . . . are available. As in the application-oriented action handle, the user may activate the appropriate action by tapping the desired command within the pop-up menu.

The preferred action handle pop-up or context menus provide a user with a consistent mechanism to interact with objects in computers that employ the teachings of the present invention. In order to maintain consistency between application, and preferred embodiment, the rules set forth in Table I are employed. A context menu is preferably divided into five logical sections. However, not all applications or objects will necessarily implement all sections:

TABLE I

First Item	The preferred first item in a context menu is a default action that occurs when the user double taps on the action handle. Thus, the user can avoid the delay associated with popping up the menu and then selecting the first item by merely double tapping the action handle or the object (or double clicking with other pointing devices).
Application Specific	The second group of commands is where application-specific information is placed. For example, in an e-mail application called Mail, this could include the commands Reply, Reply All, etc.
System	This is where all system specific functions are placed. Examples of these include Send . . . , Print . . . , Save . . . , etc.
Delete	This section contains the delete commands for the object and for ink. The delete commands include Delete Ink, Delete.
Properties	This section contains the Properties . . . selection which allows a user to select properties, attributes, or preferences.

The following Table II is an example of selected exemplary commands in a preferred order as employed in the present invention. Those skilled in the art will understand that each individual application program can utilize only the required menu selection items. Preferably, note also that the different sections are preferably separated by heavy border so as to allow readily distinguishable regions for the user to select.

TABLE II

First items	Edit . . . View
Application Specific	Undo Cut, Copy, Paste Reply, Reply All, Forward
System	Send . . . Print . . . Save . . .
Delete	Clear Ink Delete
Properties	Properties . . .

Action Handle for Variable Data Such as Text

FIG. **8**, consisting of FIGS. **8**A–**8**C, illustrates a sequence of operations and associated graphic displays that are generated in the preferred embodiment from an interaction with an action handle associated with a range **84** of selected text in an exemplary text-based application program. In the example shown in FIG. **8**, the display **18** includes a window **82** in which is displayed a string of (admittedly nonsensical) text: "This is a test of the dog that is lazy. This theory that I have, that I say which is mine, is mine. This is a test of the foxy hue."

Of the entire text string shown, only this text is selected: "test of the dog that is lazy. This theory that I have, that I say which is mine, is mine. This a test of the foxy", as shown in reverse video (white on black). This is an example of a range **84** of text that has been selected by an operation described in conjunction with FIG. **11**. The reader is referred to FIG. **11** for a discussion of the manner in which text is selected utilizing a stylus or other input device.

The range of text **84**, which is a data item, has associated therewith an action handle **40***a*, in accordance with the invention.

In the example of FIG. **8**, assume that the selected range of text **84** is to be moved by a drag-and-drop operation to another location, such as after the word "hue". The present invention supports dragging the action handle **40***a*, converting the action handle to a cursor to indicate an insertion point, and release of the action handle at a desired location, resulting in movement of the text to a new location.

In FIG. **8**A, the range of text **84** is selected, and the appropriate type of static action handle **40** is displayed, in this case a text-type static action handle **40***a* from FIG. **3**, centered within a bounding box of the text as set forth in earlier-described rules.

In FIG. **8B**, the stylus is placed into the hot spot region of the action handle **40***a* and dragged a short distance away from the original static action handle, but still within the highlighted region of selected text. Here, the computer causes generation of a dynamic action handle or drag icon **40***f*, such as the drag icon **40***f* from FIG. **4**, indicating that a move operation is being effected. Note that the particular action handle **40***f* shown in FIG. **8B** corresponds to a movement-type (but not copied-type) action handle. As previously described, the drag icon provides visual feedback to the user that a dragging operation is underway.

In FIG. **8C**, once the x-y coordinates of the stylus tip or pointing device passing outside of the selected range of text **84**, two things occur—(1) a flashing cursor or insertion point **86** is displayed underneath the stylus tip to indicate that the data item (the selected text) associated with the action handle may be "dropped" at the position indicated by the cursor **86**, and (2) a different type of drag icon **40***g* is displayed. The user indicates that he or she wishes to drop the selection at the desired point by lifting the stylus from the surface of the display screen **18** (or by clicking, if appropriate).

Note the difference between the drag icon **40***f* and **40***g*. The drag icon **40***f* has the tip of the arrow inside a small rectangular icon with a dog-eared corner, to indicate that something is being dragged. On the other hand, the drag icon **40***g* has the base of the arrow inside the rectangular, dog-eared icon, and the tip of the arrow terminating on the insertion point **86**, to indicate that the item will be put into the location pointed at by the arrow.

Those skilled in the art will understand and appreciate that the use of varying types of action handles in the present invention-static and drag icon-substantially reduces ambiguity in a GUI employing the teachings of the invention. More specifically, the second cursor type makes it easier for a user to see exactly where moved or copied text will be placed. As described in the background, there can be ambiguity in a pen-based system as to whether a move operation or a copy operation is intended. In some contexts, determination whether a move operation or copy operation is intended depends upon the destination at release of the dragged action handle. For example, a dragging operation of an action handle into something like a trash can may implicitly be interpreted as a "delete object" operation. On the other hand, a drag onto a fax machine icon or the clip box **45** shown in FIG. **1** is a copy operation.

It will now be understood that there are at least two types of drag icons utilized depending upon the destination type. If a dropping operation into a text area such as shown in FIG. **8** is intended, a standard cursor such as shown at **40***f* is utilized. This type of cursor keeps the hot-point (the tip of the arrow) so that the user can see the insertion point or cursor. In other cases, however, it may be desirable to show the actual action handle or a replica thereof being moved at the hot spot of a drag cursor. Preferably, a drag operation cursor will include an arrow so as to provide visual feedback to the user that a dragging operation is being carried out.

In FIG. **8B**, as the user drags the action handle, the cursor changes and transforms into a drag icon **40***g* with arrow comprising a large easy-to-see caret following the cursor **86**, indicating where the text will go. While the hot-spot (the tip of the arrow) is still over the selected text or source **84**, the drag icon shown at **40***f* is preferred.

As shown in FIG. **8C**, if the user moves to another portion of the document, the hot spot (tip of the arrow) culminates on the flashing insertion bar.

Referring now to FIG. **9**, if the user scrolls to another portion of the document, as by using a scrollbar (not shown in the figure), the action handle remains visible, "pinned" to the edge of the screen or document closest to the actual position of the selection. In FIG. **9**, the action handle **40** is pinned to the top border **88** of a window to indicate that the selected data item is off screen. If the user stops dragging, preferably, the action handle transforms from a drag icon such as **40***f* or **40***g* to a static action handle such as **40***a* (shown in FIG. **9**).

The user can preferably pick up the action handle and resume the drag at any time. Preferably, if the drag is resumed with the action handle "pressed" to a window border such as **88** (or a bottom border such as **89**) the text will autoscroll, that is, the text in the window scrolls upwardly or downwardly when the action handle is moved to a border of a display window, as a function of velocity of the stylus or other pointing device.

When the action handle is released by removal of the stylus from the screen, the selection appears in a new location indicated by the cursor and remains selected or highlighted.

It will understood that the scenario just described is one where the selected text is moved to another location on the screen. It should be remembered that the action handle could also have been dropped onto an application button, the clip box **45**, or other icon. If such other icons are responsive to the action indicated, appropriate action will be indicated and activated.

Turning now to FIG. **10**, if the selected range of text **84** is not big enough to contain the action handle **40**, preferably it is drawn adjacent to the selection. Note that the action handle **40** in FIG. **10** does not obscure the selected text **84**, which consists of a single word "dog" in this example. Preferably, the action handle is displayed beneath the text, in accordance with the rules described earlier herein, so that the user is not prevented from drawing gestures upon the selection or upon other text if desired (or above the selection, if there is inadequate room below).

Selection of Text With Action Handle and Cursor

Turning next to FIG. **11**, the present invention also contemplates association of an action handle with a data object such as a cursor that can be conveniently used for operations such as selecting text in a word processing type application, or selecting graphic objects such as a handwriting or a picture. The problem addressed by this aspect of the invention relates to distinguishing digital ink from commands to the computer such as selecting text. With a pointing device such as a mouse or trackball, it is known that a user can select text by simply (1) pressing a button at the beginning of the text to indicate the beginning of a selection, which usually causes display of a flashing cursor or insertion point, (2) dragging the cursor across the text, and (3) releasing the button to indicate the end of the selection. However, this press/drag/release action is ambiguous with a stylus pointing device, where it cannot be distinguished from handwriting. Therefore, this type of operation is not preferred for use with a pen, because the action of dragging with the pen is preferably reserved for drawing digital ink to make letters and/or gestures.

In the present invention, this problem is solved by associating an action handle with an insertion point in text. This allows creation of a draggable, stretchable selection. For purposes of this discussion, an "insertion point" is the place where characters typed from a keyboard are placed, typically denoted by a blinking caret or vertical bar. An insertion point can also be thought of as a selection containing zero characters.

Optionally, the present invention contemplates generation and display of the action handle only upon an initial action such as a tap by the user at a particular location in the text. This particular option requires an additional step—the initial tap to create the insertion point and associated action handle. Preferably, however, an action handle is displayed in association with a flashing insertion point continuously, so that the user will readily be able to find the insertion point (because of the size of the action handle) and will not have to remember the tap required to cause creation of the action handle.

In FIG. **11A**, therefore, there is shown an action handle **40*a*** displayed in association with a caret **90**, which is merely another type of a data object. In FIG. **11A**, no selection is indicated. The action handle **40*a*** may be considered a "selection drag button" that is centered beneath the blinking insertion point or cursor **90**.

Note also the presence of a pen icon **92**, which represents an independently displayed icon in systems that are not pen-based, or can represent the action of a stylus brought into contact with region on the display screen defining the action handle **40*a***.

In FIG. **11B**, assume that the pen (or other pointing cursor) **92** is positioned over the action handle **40*a*** and in contact with the display screen. Preferably, the pen cursor changes to an arrow **93** (in this case a "southeast" arrow.) Preferably, a southeast arrow cursor is employed rather than the northwest cursor that is often used in mouse-based systems, because a northwest cursor is too easily obscured by the pen in a user's hand.

In FIG. **11C**, the pen is pressed down and the user begins to drag the action handle **40*a*** to the right, towards the words of text "two one". In one embodiment, the action handle can disappear and the southeast arrow cursor **93** remain, as shown. Each time the insertion point crosses the midpoint of a cell in which a text letter is centered, the selection highlight adjusts accordingly to indicate selection. In the preferred embodiment, this is indicated by display of the characters in reverse video (white on black). In other words, selection proceeds similar to that employed with the press-and-hold employed with common mouse user interfaces. Note that touching the stylus outside of the action handle **40*a*** would have initiated handwriting entry.

In FIG. **11D**, when the desired selection has been established, in this example selection of the word "two" **95**, the user lifts the pen (or clicks with the mouse). Preferably, then, the cursor changes back to the pen **92**, and a static action handle **40*b*** appears centered inside the selection **95**. Optionally, the insertion point action handle **40*a*** may be retained at the end of the selection, underneath the insertion point **90**.

It will be recalled from earlier discussion that if the selection takes up more than one line, the text action handle **40*b*** should preferably be centered in the selection. Alternatively, if a selection begins near the end of one line and ends shortly after the beginning of another line, so that the selection highlighting on the two lines do not overlap, then the action handle may be centered in whichever part of the selection that is longest. Similarly, if the selection is so large that part of it is not visible in the displayed window, the action handle is preferably centered inside the visible portion of the selection.

It will also be recalled that if the user scrolls a selection by dragging an action handle to a window border, the action handle scrolls with the selection until the action handle reaches the edge of the window. If the user continues to scroll the selection out of sight, the action handle "sticks" or is pinned at the edge of the window. This feature allows the user to scroll to a distant drag/drop destination and then drag the still-visible action handle to carry out the drag/drop operation.

As shown in FIG. **11E**, in order to move the text the pen (or cursor) is positioned over the text action handle **40*b*** and the cursor changes to the appropriate dynamic drag icon, e.g. **40*c***.

As shown in FIG. **11F**, the user presses the pen down on the action handle **40*c*** and begins to drag to a new location, as at **40*d***. If desired, the static action handle can disappear, but preferably is retained, while suitable drag icon tracks the movement of the stylus. Optionally, a thin gray line or insertion cursor may be employed to track the drag icon, as shown in FIG. **11F** at **96**. Preferably, if the user lifts up the pen while the gray line is still within the highlighted selection region, the text is not moved.

As shown in FIG. **11G**, when the desired drop location has been reached, the user lifts the pen up and the selected text **95** (selected data object) appears at the new location. The text is thereby moved to the new location, formerly represented by the position of the flashing insertion point **96**. Preferably, the text remains selected, with the static icon **40** or action handle centered inside. The user can then easily move the selection again if necessary or can invoke the context menu by tapping on the action handle to act upon the selection with appropriate commands, as described.

Those skilled in the art will now understand the operation of the present invention to implement action handles in conjunction with associated data objects.

Object Oriented Programming of Action Handles

Those skilled in the art will understand that object-oriented programming languages such as C++ are preferred for preparing computer programs in accordance with the present invention, since such programming approaches facilitate the paradigm of objects consisting of action handles and associated data items. Before discussing specific objects, messages, procedures, and notifications employed in the construction of the preferred embodiment of the present invention, we will first provide a brief overview of object-oriented programming, so that the reader will understand that those skilled in the art will be enabled to make and use the present invention with the descriptions provided herein.

The basic difference between procedural and object-oriented programming is in the way that the data and action upon the data is treated. In procedural programming, data and action are two separate things. A programmer defines data structures, and then routines are written to operate on the data structures. For each data structure that is employed, a separate set of routines is required.

In object-oriented programming, actions and data are more closely related. When a programmer defines the data—which constitutes objects—he or she also defines the actions that can be taken upon such data. Instead of writing a set of routines that do something to a separate set of data, the programmer creates a set of objects that interact with each other by passing messages back and forth.

In "object" is an entity that contains some data and an associated set of actions that operate on the data. The data can be extracted from the object and sent to another object, for example when the data is to be stored in a file. To make an object perform one of its actions, the object is sent a message. For example, in the present invention the programmer will create an object that represents the action handle, and similarly will create objects that represent data items such as an address, a selected range of text, a insertion point, or the like. The data associated with the object such as the action handle will contain locations of the boundaries of the icons and the definition of the hot spots, a bit map of the graphic appearance of the action handles, the present coordinates of the action handles, etc., as well as indication that various actions can be taken upon appropriate activation. For example, to cause an action handle to pop up a context menu, an action handle object is sent a "tap" message.

For the time being, the manner in which objects are sent messages is beyond the scope of the present discussion. Such operations are typically within the purview of the operating system or of the compiler that is employed to generate program code.

Contrast the foregoing with the manner in which an action handle is programmed with procedural programming. First, the programmer must define a data record that represents the boundaries of the icon relative to the display screen, and must translate these local boundaries to the global coordinate system associated with the display screen. The programmer will then have to write different routines for the different actions that could be performed when a user interacts with the action handle, namely, one to cause a pop-up menu to be generated upon a tap, one to cause generation of an "inside selection" drag icon as the action handle is dragged within the range of selected text, and a different routine to cause display of an "out of selection" drag icon as the action handle is dragged outside of the selected text. These routines would have to be rewritten for each different type of action handle and the associated actions that could be carried out with respect to that particular type of action handle.

A particular advantage of object-oriented programming over procedural programming is that a programmer can keep the routines that operate on a data structure together with the data structure that they are operating on. This is called "encapsulation".

A more significant advantage in object-oriented programming is that objects can inherit data and behavior from other, similar objects. Every object belongs to a "class", which defines the implementation of a particular kind of object. A class describes the object's data and the messages to which it responds.

Classes are somewhat related to record declarations in conventional programming languages. A class may be thought of as a template for creating objects. The class describes an object's data and the messages to which it responds. A particular object is called an instance of a class. The programmer defines data for the class the same way that fields of a record are defined. With classes and objects, though, fields are called "instance variables". Each instance of a class, that is each object, has its own instance variables, just as each variable of a record type has the same fields.

When a message is sent to an object, the object invokes a routine that implements that message. These routines are called "methods". A class definition includes method implementations. Method implementations are sequences of computer program steps that are carried out in response to a particular message.

It is important to keep in mind that a message and a method are not the same thing. A message is sent to an object. However, an object responds to a message by carrying out a method.

Turn next to FIG. **12** for an illustration of these object-oriented concepts. This figure illustrates an exemplary class definition **100** for an object type, "Rectangle". Assume for purposes of this discussion that a Rectangle object is a computer program that is operative to draw a rectangle on the display screen in response to being sent a "Draw" message. For this exemplary Rectangle class, there are instance variables consisting of the Upperleft Corner, Upperright Corner, Lowerleft Corner, and Lowerright Corner. There might also be an instance variable indicative of the thickness of the line to be drawn between the corners to define the rectangle, Linewidth. There might also be an instance variable, Fillpattern, indicative of a fill pattern to be filled or drawn inside the boundaries of a rectangle.

Three exemplary messages are shown for the rectangle class: Draw, Erase, and Move. The Draw message causes invocation of a method for drawing a line from point to point, where the points are represented by the instance variables Upperleft Corner, Upperright Corner, Lowerleft Corner, and Lowerright Corner. Similarly, an Erase message causes the lines to be erased. Finally, a Move message causes the corner points to be translated or offset to another location. After a Move operation, a new Draw message might be sent, so that the rectangle appears on the screen to have moved from one position on the display to another.

The concept of "notifications" in object oriented programming should also be kept in mind. Like programming of subroutines, the programming of objects will often result in the generation of return results or messages that must be delivered back to an object that initiated an operation by sending a message. These return results or messages may also be called notifications. For example, a calculating object that is sent an "Add" message with instance variables "2" and "3" might return a notification of "5", which is the result of the addition operation 2+3. Similarly, a drawing object (Rectangle) that is sent a Draw message together with coordinates that are outside the range of the display screen might return a notification (e.g. an error notification) that the operation cannot be completed as requested. Specific notification messages utilized in the present invention are discussed in greater detail below.

Inheritance and Polymorphism

Programmers utilizing an object-oriented programming environment can define a class in terms of an existing or higher up class. The existing class is called a superclass, and a new class may be called a subclass. A class without a superclass is said to be a root class. A subclass inherits all the instance variables and methods of an associated superclass. Subclasses can define additional instance variables and methods. Subclasses can also override methods defined by the superclass, so as to provide different operations in different contexts.

Overriding a method means that the subclass responds to the same message as its associated superclass, but that the subclass object uses its own methods to respond to the message.

For example, and looking at FIG. **13**, suppose the programmer wants to create a class to represent icons displayed on the display screen. This is shown as the Class: Icon **105**. The instance variables might include a Bitmap (which is associated with a graphic image displayed on the screen), a Location (which is the x-y coordinates of the icon relative to the display screen), and an Underlying Object (which is another object that can be accessed or controlled by interacting with the icon, for example, an application program that is activated if the icon is clicked on).

Exemplary messages that could be sent to a member (or instance) of the Icon class might include Select, Activate, ShowMenu, Drag, Drop, Delete, and DoubleTap. The programmer defines various steps of actions comprising computer program steps to correspond to each of these methods. For example, in response to a Select message, the programmer may include program steps that causes the display of the Bitmap in reverse video on the display screen. Similarly, in response to a Activate message, the programmer might provide commands to send a Activate message to an Underlying Object, and then display the Bitmap in reverse video and normal video in rapid succession to indicate visually that an activation process is underway. A ShowMenu message might cause the display of a context menu. A Drag message might cause translation or offset of the location of the Bitmap relative to the display screen. A Drop message might cause an action to take place that is appropriate for a location at which an object is dropped, e.g. merely redrawing the icon at another screen location, or, if an object's icon is dropped atop a printer icon, printing that object's data. A Delete message might cause deletion of the Bitmap from the display screen, as well as of the Underlying Object (which would result in freeing up memory). A DoubleTap message might cause execution of the first command on a command list associated with the icon. Other messages and methods will occur to those skilled in the art.

Still referring to FIG. **13**, after a programmer has been provided with or created program code for an Icon class that carries out methods corresponding to messages, he or she can then use the Icon class to create subclasses. Consider for example the Icon subclasses in FIG. **13**, Program Icon **108** and Action Handle Icon **110**. Both of these subclasses inherit all the instant variables from the Icon class. Each class, however, may define new instance variables and each subclass can override certain methods. For example, the Program Icon **108** may include a Memory Allocation instance variable that stores data corresponding to the desired memory utilized by the Underlying Object, which is another computer program.

Similarly, the Action Handle Icon subclass **110** will include additional Bitmap instance variables, e.g. Bitmap**1**, Bitmap**2**, Bitmap**3**, corresponding to different types of static action handles and drag icons that are displayed in response to various conditions in accordance with the invention.

Each of these subclasses Program Icon **108** and Action Handle Icon **110** responds differently to an Activate Message. The Program Icon responds to an Activate Message by loading the associated program (the Underlying Object), while the Action Handle Icon responds to an Activate Message by popping up the context menu, as has been described elsewhere herein.

As long as the objects are members of any of the associated Icon classes, they can be sent messages that are common to them all and the object will respond properly. The ability to send the same message to objects of different classes or subclasses is called "polymorphism". If the programmer needs to create a different kind of Icon, all he or she needs to do is define a subclass of Icon (by copying or duplicating the program code previously written for the Icon class of objects) and override such methods and provide additional instance variables as are required.

With the foregoing background in mind, it will now be appreciated that the present invention is preferably made with object-oriented programming techniques. A complete discussion of all the possible objects that could be implemented in a computer program employing the teachings of the invention is beyond the scope of the invention and unnecessary for the skilled artisan. To enable the skilled artisan to carry out the invention, we will concentrate the remaining discussion on the various objects that will be constructed in accordance with the invention.

FIG. **14** illustrates exemplary classes and relationships between classes that are provided in the present invention to implement action handles in accordance with the present invention. Discussion will not be provided of classes that are shown as the operating system object **120**, the event handler object **125**, a program application denominated HH APPLICATION **130**, and a window object HHWINDOW **135**. These and other objects are created by the programmer to write computer program code for a particular application. For purposes of the present invention, it will be understood that:

- an event handler object **125** receives user input such as a tap, click, or drag, and translates the user input into appropriate messages such as Tap, Click, or Drag, and provides these messages to the operating system **120**.
- the operating system object **120** coordinates the resources of the computer system such as the event handler, disk drives, display screen, etc. It receives messages from various underlying objects, and passes messages back and forth between objects as appropriate.
- an HHAPPLICATION object **130** is an object that implements methods for a particular user purpose or application, for example, a Notes HHAPPLICATION object would handle storage, retrieval, and display of notes entered by a user.
- an HHWINDOW object **135** is an object that handles display of data in a window on a display screen. Intuitively, a "window" object represented by a member of the HHWINDOW class corresponds to any type of window that can be displayed, in the abstract, on the display screen.

an HHPANE object **138** is a subclass of HHWINDOW object that displays a selected subset of data on a display screen in a smaller "pane", e.g. merely a range of selected text, or a user control button. A member of the subclass HHPANE can be a subwindow such as a dialog box, a button, or other species of a window that can be displayed on a display screen.

a HHBUTTON object **140** is a subclass of HHPANE object that comprises a user control. An action handle is an HHBUTTON type object.

a HHLISTBOX object **150** is a subclass of HHPANE object that comprises a displayed list of data items. A HHLISTBOX object has a pop-up or context menu that is displayed in response to a tap on an action handle.

From the foregoing, it will be understood that action handles in accordance with the present invention are types of control buttons, HHBUTTON, and these are subclasses of the HHPANE class **138**. The discussion which follows provides further information on the nature of the instance variables, messages, and methods required to implement objects that constitute action handles and associated data.

TABLE III below illustrates various events that are handled by the event handler object **125** (FIG. **14**), and the messages generated or actions taken. It will be understood that a "click/tap" event comprises a tap or a mouse down action, i.e., the contacting of a stylus with the display screen or the depression of a mouse button. A "drag" event in a pen-based system comprises a contacting of the stylus with the display screen at a first or starting location on the display screen, and movement of the stylus along a path on the display screen while maintaining the stylus in contact with the display screen, followed by lifting the stylus at a desired second or ending location. For a mouse or similar pointing device, a "drag" comprises a mouse down at a first or starting cursor location on the display screen, and movement of the cursor along a path on the display screen while maintaining the mouse button down, followed by releasing the mouse button at a desired second or ending location.

TABLE III

Event	Location	Action
click/tap	in action handle	pop up context menu- send
drag	in action handle	relocate data object and action handle to new display location at mouse up
click/tap	in data object	select data object for text editing or other action appropriate for particular type of data object
drag	in data object	no effect, or object defined (e.g. handwriting)
click/tap	in window in which action handle and data object appear	activate window if beneath; otherwise nothing
drag	in window in which action handle and data object appear	activate window if beneath; otherwise nothing

It will be further understood that the click/tap and drag actions for the window in which an action handle and data object appear are the same actions as those taken in the known WINDOWS™ graphic user interface.

TABLES IV–VI below illustrate various aspects of the HHBUTTON class of objects **140** and HHLISTBOX class of objects **150** that are programmed in accordance with the teachings herein, for implementing action handles in accordance with the present invention. Those skilled in the art will understand that the class definitions provided in the following tables, which should be self explanatory after the foregoing discussion on object-oriented programming, provide an enabling disclosure of the class, instance variables, methods, and notifications unique to the HHBUTTON and HHLISTBOX classes, that differ from classes of a higher hierarchy such as HHPANE and HHWINDOW, for preparing computer program code in accordance with the invention. The tables describe particular display styles for associated graphic images associated with instances of the object, a general description of the class and its functions, messages provided to members of the class and corresponding methods carried out by members of the class, and notification messages provided in response to execution of program code to effectuate the methods described hereinabove.

TABLE IV

HHBUTTON Class Definition	
Class HHBUTTON	
Instance Variables: N/A	
Messages & Notifications	Listed below
Styles:	
HHBS_DRAGICONBMP	Button is an action handle
HHBS_DRAGICONBRD	Button is an action handle with a bitmap with a raised border
HHBS_DRAGICONBTN	Button is an action handle bitmap with a border.

Description This is a control object that designates a small rectangular "child" window that represents a button the user can activate by tapping or other user event. There are different types of buttons for different uses, each considered a subclass, e.g. action handles or drag icons.

Drag icons are used to implement a drag handle for data that does not have a drag handle implemented by a control. For example, list boxes, edit controls and note controls all support drag handles, but one could use a drag icon to associate a drag handle with data in a group of controls.

Most push buttons resemble the raised-button look of a numeric keypad. They represent something the user can tap to perform an action or set a state. Regular push buttons are displayed in a depressed position when the user taps it and as in a raised position when released. In contrast, a sticky button remains in the depressed position until the next click.

Messages:	Methods:
HHBM_GETBITMAP	Retrieves the handle to the bitmap displayed on the button
HHBM_SETBITMAP	Specifies the handle of the bitmap used to display the bitmap
HHBM_GETDRAGICONCMD	Identifies the command invoked when a user taps on the button; Generate the message WM_DRAGDROPREQUEST
HHBM_GETDRAGICONCM	Retrieves a list of

-continued

DLST	commands (HCMDLST) to be displayed in a pop up context menu
HHBM_SETDRAGICONCM DLST	Specifies the list of commands (HCMDLST) for the action handle that appears when a user tapes the button

Notifications: The following describes the notification messages that can be sent by a HHBUTTON object to its parent via a WM_COMMAND message

HHBN_CLICKED	Sent when the user taps the action handle
HHBN_DBLCLK	Indicates that the user double clicked or double tapped on an action handle
HHBN_DRAGICONCMD	Sent when the user taps on an action handle pop up menu item. The associated application program must then send the message HHBM_GETDRAGICONCMD to identify the command invoked.
HHBN_DRAGICONCMDLST	Allows the application to set or change the command list prior to showing it. This message is sent just before displaying the command list.
HHBN_TIMEOUT	Sent to indicate a time out operation, if necessary
WM_DRAGDROPREQUEST	Notifies the parent object of action handle that a dragging operation is desired, and to handle a drag and drop operation.

TABLE V

HHLISTBOX Class Definition

Class	HHLISTBOX
Instance Variables:	N/A
Messages & Notifications	Listed Below
Description	The HHListbox control is used to list information for applications, such Address, ToDo, and Calendar.

The HHLISTBOX class of objects is the class that controls display of a pop-up or context menu of selectable items or commands.

There are two types of selection modes depending on whether the HHLS_EXTENDEDSEL style is set: single-selection mode and extended-selection mode.

In single-selection mode, a pen down event on a list box item removes any previous selection and displays the item on which the pen is positioned as highlighted. The control tracks the pen until the pen is lifted and the new item is selected. That is, only a single item is ever selected at any one time.

When the list box control is in extended-selection mode, placing the pen down on an item removes any previous selection and highlights the item. When the pen is lifted, all items between the item on which the pen was placed and the item on which the pen was lifted become selected (not all items that have been touched).

List box controls with the HHLS_DRAGABLE style support action handles. The application can set a default handheld icon, a handheld icon for a multiple selection and specific handheld icons for each list box item. If no handheld icon is set for an item, the default icon will be used. Though the list box will handle the drawing of the drag icon, the parent has the responsibility of responding to the WM_DRAGDROPREQUEST message in order to support drag and drop from a listbox.

Action handles will be on the left of all lines in a list box item.

Styles The following describes the HHListbox control styles (in the HHLISTBOX class) specified in the dwStyle parameter of the CreateWindow function.

Style	Description
HHLS_DRAGABLE	Specifies that the list box has action handles associated therewith.
HHLS_EXTENDEDSEL	Allows a range of items to be selected.
HHLS_NOTIFY	Sends an input message to the parent window whenever the user taps or double-clicks a string.

Messages The following describes the messages that can be sent to an HHListBox control. Again, action handles are synonymous with drag icon.

Message	Method
HHLB_GETDEFAULTDRAGICON	Retrieves information about the default drag handle.
HHLB_SETDEFAULTDRAGION	Specifies the default drag icon attributes
HHLB_GETDRAGICONCMD	Retrieves the identity of the command chosen from a command list.
HHLB-GETDRAGICONCOL	Retrieves the width, in pixels, of the drag icon column.
HHLB_SETGRAGICONCOL	Specifies the size, in pixels, of the left column where drag icons are displayed.
HHLB_GETITEMDRAGICON	Retrieves the information about the list box drag icon.
HHLB_SETITEMDRAGICON	Specifies the drag icon information of the list box item.

Notifications The following describes the actions of the notification Messages sent by the HHListBox concontrol.

Notification Messages	Action
HHLBN_DRAGICONCMDLST	Sent when the listbox is about to display its command list.
HHLBN_DRAGICONCMD	Sent when the listbox has received the WM_COMMAND message from the command list item.
HHLBN_ERRSPACE	Sent when list box cannot allocate enough memory to meet a specific request.

The following TABLE V contains a reference of exemplary control messages that are employed in the preferred embodiment of the present invention.

TABLE VI

Control Messages
HHBM_GETBITMAP

```
HHBM_GETBITMAP
wParam = OL; /* not used, must be zero */
lParam = (LPARAM) MAKELPARAM(fPressed); /*
   pressed or normal */
```

An application sends an HHBM_GETBITMAP message to retrieve the handle to the bitmap being displayed on the button. Applies only to buttons with the style HHBS PUSHBUTTON. HHBS_VERTICALBUTTON. HHBS_STICKYBUTTON. HHBS_AUTOSTICKYBUTTON. HHBS_DRAGICONBTN and HHBS_DRAGICONBMP.

Parameters	Pressed
	For HHBS_DRAGICONBMP and HHBS_DRAGICONBTN styles, determines whether to retrieve the bitmap used for the pressed or non-pressed version of the button. Zero returns the non-pressed version.
Return Value	The return value is the handle to the bitmap if one exists for the button, else NULL The return value is always NULL for buttons that do not have the right style. For HHBS_DRAGICONBMP and HHBS_DRAGICONBTN, if the low-order word of lParam is non-zero, it will retrieve the presses version of the bitmap. HHBM_SETBITMAP does the same to set it.

HHBM_GETDRAGICONCMD

```
HHBM_GETDRAGICONCMD
wParam = 0;          /*not used, must be zero */
lParam = 0L;         /*not used, must be zero */
```

This message identifies the command invoked when the user taps on a control with the button style HHBS_DRAGICONBTN, HHBS_DRAGICONBMP, or HHBS_DRAGICONBRD. An application sends an HHBM_GETDRAGICONCMD message during the processing of the HHBN_DRAGICONCMD notification to retrieve the command id of the command list item picked by the user.

Parameters	This message has no parameters
Return	Value Returns the command ID, or zero.

HHBM_GETDRAGICONCMDLST

```
HHBM_GETDRAGICONCMDLST
wParam = 0;          /*not used, must be zero */
lParam = 0L;         /*not used, must be zero */
```

An application sends HHBM_SETDRAGICONCMDLST to a button with style HHBS_DRAGICONBTN, HHBS_DRAGICONBMP, or HHBS_DRAGICONBRD to retrieve the handle of the command list (i.e., context or pop up menu) it will show when the user taps or clicks on it.

Parameters	This message has no parameters.
Return Value	Returns a handle to the command list, or NULL if none is set. ff the style is invalid, NULL is always returned.

HHBM_SETBITMAP

```
HHBM_SETBITMAP
wParam = (WPARAM) hBitmap;
        /* handle to bitmap */
lParam = (LPARAM) MAKELPARAM(fPressed);
        /* pressed or normal bitmap */
```

An application sends a HHBM_SETBITMAP message to set the handle of the bitmap to be used when displaying the action handle. For color bitmaps, the blue bits will appear transparent. Color bitmaps are device dependent.

The bitmap will be centered if there is no text, else it will appear to the left of the text. This applies only to buttons with the styles HHBS_VERTICALBUTTON,

HHBS_STICKYBUTTON, HHBS_AUTOSTICKYBUTTON, HHBS_DRAGICONBTN, or HHBS_DRAGICONBMP.

Parameters	hBitmap Value of wParam. Handle to the bitmap to be set fPressed For HHBS_DRAGICONBTN, or HHBS_DRAGICONBMP, and HHBS_DRAGICONBRD styles, determines whether to set the bitmap used for the pressed or non-pressed version of the button. Zero sets the non-pressed version.
Return Value	The return value is TRUE for a successful setting, else FALSE.

HHBM SETDRAGICONCMDLST

```
HHBM_SETDRAGICONCMDLST
wParam = hcmdlst;
lParam = 0L;         /*not used, must be zero */
```

An application sends HHBM_SETDRAGICONCMDLST to a button with style HHBS_DRAGICONBTN or HHBS_DRAGICONBMP to set the handle of the command list it will show when the user taps or clicks on it.

Parameters	hcmdlst Value of wParam. Specifies the handle of the command list to be used
Return Value	Returns a nonzero if set, or a 0 for an invalid style.

HHEM_CHECKDRAGICON

```
HHBM_CHECKDRAGICON
wParam = 0;          /*not used, must be zero */
lParam = 0L;         /*not used, must be zero */
```

An application sends an HHEM_CHECKDRAGICON message to cause the control to update its drag icon if needed.

Parameters	This message has no parameters
Return Value	None

HHLB_GETDEFAULTDRAGICON

```
HHBM_GETDEFAULTDRAGICON
wParam = 0;              /*not used, must be zero */
lParam = (LPARAM)lplbdragicon;
                         /*ptr. to lbdragicon structure */
```

An application sends the HHLB_GETDEFAULTDRAGICON message to retrieve the drag icon information for the default drag icons for the list box. Sending this message causes the lbdragicon structure specified by the IParam to be filled

Parameters	Iplbdragicon Specifies a long pointer to a LBDRAGICON structure, to fill with default drag icon information.

	typedef struct tagLBDRAGICON
	{
	HBITMAP hbmUp;
	HBITMAP hbmDown;
	LBDRAGICON;
	}

Return Value	Returns HHLB_OKAY if the structure was filled in with the default drag icon information. It returns HHLB_ERR if an error occurs or if the listbox is without the style HHLS_DRAGABLE.

HHLB_GETDRAGICONCMD

```
HHBM_GETDRAGICONCMD
wParam = 0;              /*not used, must be zero */
lParam = 0L;             /*not used, must be zero */
```

An application sends an HHLB_GETDRAGICONCMD message to retrieve the command id of the item selected by the user from the command list. This message is only valid during processing of the HHLBN_DRAGICONCMD notification

Parameters	This message has no parameters.
Return Value	Returns the ID of the command list item, or HHLB_ERR in a listbox without the style HHLS_DRAGABLE.

HHLB_GETDRAGICONCOL

```
HHBM_GETDRAGICONCOL
wParam = 0;              /*not used, must be zero */
lParam = 0L;             /*not used, must be zero */
```

An application sends an HHLB_GETDRAGICONCOL message to retrieve the size, in pixels, of the column that displays icons on the left side of a listbox.

Parameters	This message has no parameters.
Return Value	Returns the width of the column on the left side of the list box to show drag icons in a listbox with the style HHLS_DRAGABLE, or else HHLB_ERR.

HHLB_GETITEMDRAGICON

-continued

```
HHBM_GETITEMDRAGICON
wParam = index;     /*not used to item */
lParam = (LPARAM)lplbdragicon;  /*pnt. to lbdragicon
structure */
```

An application sends the HHLB_GETITEMDRAGICON message to retrieve the handheld icon information of the listbox item in a listbox with the HHLS_DRAGGABLE style

Parameters	index Specifies the zero-based index of the item's handle to be retrieved. Iplbdragicon Specifies a long pointer to a LBDRAGICON structure to be filled.

	typedef struct tagLBDRAGICON
	{
	HBITMAP hbmUp;
	HBITMAP hbmDown;
	LBDRAGICON;
	}

Return Value	Returns the height, in pixels, of a listbox item.

HHLB_SETDEFAULTDRAGICON

```
HHBM_SETDEFAULTDRAGICON
wParam = 0;              /*not used, must be zero */
lParam = (LPARAM)l;plbdragicon;
                         /*ptr. to lbdragicon structure */
```

An application sends the HHLB_SETDEFAULTDRAGICON message to set the default drag icons used by the list box to the bitmaps in the specified Ibdragicon structure.

Parameters	Iplbdragicon Long pointer to a LBDRAGICON structure or NULL for the item to use the default icon.

	typedef struct tagLBDRAGICON
	{
	HBITMAP hbmUp;
	LBDRAGICON;
	}

Return Value	Returns HHLB_OKAY if default drag icon 15 set, or else HHLB_ERR if an error occurred in a listbox without the style HHLS_DRAGABLE.

HHLB_SETDRAGICONCOL

```
HHLB_SETDRAGICONCOL
wParam = (WPARAM)colwidth; /* new width of
                         drag icon column */
lParam = 0L;             /* not used, must be zero */
```

An application sends an HHLB_SETDRAGICONCOL message to set the size, in pixels, of the column that displays drag icons on the left side of a listbox.

Parameters	colwidth Value of wParam.. The width, in pixels, of the new width of the drag icon column.
Return Value	Returns the width of the column on the left side of the listbox to show drag icons in a listbox with the style HHLS_DRAGABLE, or else HHLB_ERR.

-continued

HHLB_SETITEMDRAGICON

```
HHLB_SETITEDRAGICON
wParam = index;              /* index to item */
lParam = (LPARAM)lpLbdragicon;
                             /* ptr to lbdragicon struct */
```

An application sends the HHLB_SETITEMDRAGICON message to set the handheld icon information of a list box item's icon. The list box must have the style HHLS_DRAGGABLE set.

Parameters	index Specifies the zero-based index of the item

	typedef struct tagLBDRAGICON
	{
	HBITMAP hbmUp;
	HBITMAP hbmDown;
	LBDRAGICON;
	}

Return Value	The return value is 0 or HHLB_ERR if an error occurred

Notification Messages

This section contains a reference of notification messages that are employed in the preferred embodiment of the present invention.

HHBN_DRAGICONCMD

The HHBN_DRAGICONCMD notification message is sent by buttons with the button style HHBS_DRAGICONBTN or HHBS_DRAGICONBMP. It indicates that the control has just received a WM_COMMAND from its command list indicating that the user has just picked a command list item. The application can query which command item was picked by sending the HHBM_GETDRAGICONCMD message.

Parameters	index Specifies the zero-based index of the item lplbdragicon Long pointer to a LBDRAGICON structure or NULL for the item to use the default icon.

	typedef struct tagLBDRAGICON
	{
	HBITMAP hbmUp;
	HBITMAP hbmDown;
	} LBDRAGICON;

Return Value	The return value is 0 or HHLB_ERR if an error occurred.

HHBN_DRAGICONCMDLST

The HHBN_DRAGICONCMDLST notification message is sent to the application to allow it to set or change the command list prior to showing it This message is sent by a control with button style HHBS_DRAGICONBTN or HHBS_DRAGICONBMP.

Parameters	wParam Specifies the idenfifier of the button lParam Specifies the handle of the control box in the low-order word, and specifies the HHBN_DRAGICONCMDLST notification message in the high-order word.

HHLBN_DRAGICONCMD

This notification message is sent by a listbox with the style HHLS_DRAGABLE. It indicates that the user has picked an item from the listbox's command list. The application can query the command identity of the selected command list item by sending an HHLB_GETDRAGICONCMD.

Parameters	wParam Specifies the identifier of the button lParam Specifies the handle of the list box in the low-order word, and specifies the HHBN_DRAGICONCMD notification message in the high-order word.

HHLBN_DRAGICONCMDLST

This notification message is sent by a listbox with the style HHLS_DRAGABLE. It indicates that the listbox is about to display its command list. The application can find out what listbox items are associated with the active drag icon by sending an HHLB_GETDRAGICONITEMS message to the listbox. If necessary, modify the command list associated with the list box by using the HHLB_GETCMDLST and HHLB_SETCMDLST messages.

Parameters	wParam Specifies the idenfifier of the button lParam Specifies the handle of the list box in the low-order word, and specifies the HHBN_DRAGICONCMDLST notification message in the high-order word.

It will now be understood that the present invention has been described in relation to particular embodiments which are intended in all respects to be illustrative rather than restrictive. Alternative embodiments will become apparent to those skilled in the art to which the present invention pertains without departing from its spirit and scope. Accordingly, the scope of the present invention is defined by the appended claims rather than by the foregoing description.

What is claimed is:

1. In a computer controlled display system having a display coupled to a central processing unit (CPU) for displaying images, a method for displaying and controlling actions relating to a data object displayed on the display, comprising the steps of:

generating a graphic image associated with the data object on the display;

generating a graphic image of an action handle proximate to the data object;

providing an activatable region on the display associated with the action handle graphic image;

in response to the user's positioning of a pointer on the display with a pointing control device operatively associated with the CPU, providing a signal to the CPU upon correspondence of the pointer with the activatable region of the action handle;

in response to a signal indicative of a first type of interaction between the pointer and the activatable region associated with the action handle, displaying at least one selectable command relating to the data object in a context menu including the at least one selectable command; and

in response to signal indicative of a second type of interaction between the pointer and the activatable region associated with the action handle, moving the graphic image of the data object on the display.

2. The method of claim **1**, wherein the first type of interaction is a tap.

3. The method of claim **1**, wherein the second type of interaction is a drag.

4. The method of claim **1**, wherein the step of displaying at least one selectable command comprises:

displaying a plurality of selectable commands in the context menu.

5. The method of claim **1**, further comprising the steps of:

providing an activatable second region in the context menu on the display associated with the selectable command; and

in response to interaction between the second region and the pointer, executing a computer command corresponding to the selectable command.

6. The method of claim **5**, wherein the interaction between the pointer and the second region comprises a tap.

7. The method of claim **1**, wherein the step of moving the graphic image of the data object on the display comprises a drag-and-drop operation.

8. The method of claim **1**, wherein the step of moving the graphic image of the data object on the display comprises:

moving the graphic image of the action handle on the display during a dragging operation from a first position to a second position on the display;

in response to a release command provided at the second position on the display, moving the graphic image of the data object to a new position on the display associated with the second position.

9. The method of claim **1**, wherein the data object is a data item associated with a predetermined application program.

10. The method of claim **9**, wherein the data item is a selected string of text associated with a text processing application program.

11. The method of claim **1**, wherein the display system includes a direct pointing device for providing signals to the CPU.

12. A computer controlled display system, comprising:

a central processing unit (CPU);

a memory coupled to said CPU for storing programs and data;

a graphic display coupled to said CPU for displaying information;

a pointing device coupled to said CPU for allowing an operator to point to items displayed on said display and for providing signals to said CPU indicative of predetermined actions;

said CPU being operative to:

store data corresponding to a data object in said memory;

display a data object graphic image associated with said data object on said display;

display an action handle proximate to said data object graphic image on said display, said action handle comprising a predetermined activatable region on said display associated with said action handle;

in response to a signal from said pointing device indicative of a first type of interaction between said pointing device and said activatable region of said action handle, display at least one selectable command operatively associated with said data object in a context menu including the at least one selectable command; and

in response to a signal from said pointing device indicative of a second type of interaction between said pointing device and said activatable region of said action handle, reposition said data object graphic image on said display.

13. The system of claim **12**, wherein said first type of interaction between said pointing device and said activatable region comprises a tap.

14. The system of claim **12**, wherein said second type of interaction is a drag.

15. The system of claim **12**, wherein the

context menu on the display includes a plurality of selectable commands.

16. The system of claim **12**, wherein said CPU is further operative for:

providing an activatable second region on the display associated with said selectable command; and

in response to interaction between the second region and said pointing device, executing instructions corresponding to said selectable command.

17. The system of claim **16**, wherein said interaction between said pointing device and said activatable second region comprises a tap.

18. The system of claim **12**, wherein said CPU operation of repositioning said data object graphic image on said display comprises a drag-and-drop operation.

19. The system of claim **12**, wherein said CPU operation of repositioning said data object graphic image on said display comprises:

moving said graphic image of said action handle on said display during a dragging operation from a first position to a second position on said display;

in response to a release command provided at said second position, moving said graphic image of said data object to a new position on said display associated with said second position.

20. The system of claim **12**, wherein said data object is a data item associated with a predetermined application program.

21. The method of claim **20**, wherein said data item is a selected string of text associated with a text processing application program.

22. The method of claim **20**, wherein said data item comprises a calendar entry associated with a calendar application program.

23. In a computer controlled display system having a display coupled to a central processing unit (CPU) for displaying images, a method for displaying and controlling actions relating to a selected text string displayed on the display, comprising the steps of:

generating a graphic image of a string of text on the display;

in response to selection of predetermined selected text within the string of text, generating a graphic image of an action handle in a region proximate to the predetermined selected text;

providing an activatable region on the display associated with the action handle;

in response to a user's positioning of a pointer on the display to the activatable region associated with the action handle using a pointing control device operatively associated with the CPU, providing signals to the CPU indicative of correspondence of the pointer with the action handle;

in response to a signal indicative of a first type of interaction between the pointer and the activatable region associated with the action handle, displaying at least one selectable command relating to the predetermined selected text in a context menu including the at least one selectable command; and

in response to a signal indicative of a second type of interaction between the pointer and the activatable region associated with the action handle, moving the predetermined selected text on the display.

24. The method of claim **23**, wherein the step of displaying at least one selectable command comprises:

displaying a plurality of selectable commands in the context menu.

25. The method of claim **24**, wherein the context menu comprises a list of a predetermined plurality of selectable commands, and wherein the selectable commands at the beginning of the list are of a first, application specific type and wherein the commands at the end of the list are of a second, operating system specific type.

26. The method of claim **23**, further comprising the steps of:

providing an activatable second region on the display associated with the selectable command; and

in response to interaction between the second region and the pointer, executing a computer command corresponding to a particular selectable command.

27. The method of claim **26**, wherein the interaction between the pointer and the second region comprises a tap.

28. The method of claim **23**, wherein the step of moving the predetermined selected text on the display comprises a drag-and-drop operation.

29. The method of claim **23**, wherein the step of moving the predetermined selected text on the display comprises:

moving the action handle on the display during a dragging operation from a first position to a second position on the display;

in response to a release command provided at the second position on the display, moving the predetermined selected text to a new position on the display associated with the second position.

30. The method of claim **23**, wherein the string of text is displayed on the display in a scrollable window, and further comprising the steps of

in response to a signal indicative of the second type of interaction between the pointer and the action handle and movement of the action handle to a boundary of the scrollable window, scrolling the text in the scrollable window; and

continuing to display the action handle pinned to the boundary of the window during scrolling of the text.

* * * * *

Exhibit X

To

Joint Claim Chart

International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

H.264

(03/2010)

SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS

Infrastructure of audiovisual services – Coding of moving video

Advanced video coding for generic audiovisual services

Recommendation ITU-T H.264

ITU-T H-SERIES RECOMMENDATIONS

AUDIOVISUAL AND MULTIMEDIA SYSTEMS

CHARACTERISTICS OF VISUAL TELEPHONE SYSTEMS	H.100–H.199
INFRASTRUCTURE OF AUDIOVISUAL SERVICES	
General	H.200–H.219
Transmission multiplexing and synchronization	H.220–H.229
Systems aspects	H.230–H.239
Communication procedures	H.240–H.259
Coding of moving video	**H.260–H.279**
Related systems aspects	H.280–H.299
Systems and terminal equipment for audiovisual services	H.300–H.349
Directory services architecture for audiovisual and multimedia services	H.350–H.359
Quality of service architecture for audiovisual and multimedia services	H.360–H.369
Supplementary services for multimedia	H.450–H.499
MOBILITY AND COLLABORATION PROCEDURES	
Overview of Mobility and Collaboration, definitions, protocols and procedures	H.500–H.509
Mobility for H-Series multimedia systems and services	H.510–H.519
Mobile multimedia collaboration applications and services	H.520–H.529
Security for mobile multimedia systems and services	H.530–H.539
Security for mobile multimedia collaboration applications and services	H.540–H.549
Mobility interworking procedures	H.550–H.559
Mobile multimedia collaboration inter-working procedures	H.560–H.569
BROADBAND, TRIPLE-PLAY AND ADVANCED MULTIMEDIA SERVICES	
Broadband multimedia services over VDSL	H.610–H.619
Advanced multimedia services and applications	H.620–H.629
IPTV MULTIMEDIA SERVICES AND APPLICATIONS FOR IPTV	
General aspects	H.700–H.719
IPTV terminal devices	H.720–H.729
IPTV middleware	H.730–H.739
IPTV application event handling	H.740–H.749
IPTV metadata	H.750–H.759
IPTV multimedia application frameworks	H.760–H.769
IPTV service discovery up to consumption	H.770–H.779

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T H.264

Advanced video coding for generic audiovisual services

Summary

This Recommendation | International Standard represents an evolution of the existing video coding standards (H.261, H.262, and H.263) and it was developed in response to the growing need for higher compression of moving pictures for various applications such as videoconferencing, digital storage media, television broadcasting, Internet streaming, and communication. It is also designed to enable the use of the coded video representation in a flexible manner for a wide variety of network environments. The use of this Recommendation | International Standard allows motion video to be manipulated as a form of computer data and to be stored on various storage media, transmitted and received over existing and future networks and distributed on existing and future broadcasting channels.

The revision approved 2005-03 contained modifications of the video coding standard to add four new profiles, referred to as the High, High 10, High 4:2:2, and High 4:4:4 profiles, to improve video quality capability and to extend the range of applications addressed by the standard (for example, by including support for a greater range of picture sample precision and higher-resolution chroma formats). Additionally, a definition of new types of supplemental data was specified to further broaden the applicability of the video coding standard. Finally, a number of corrections to errors in the published text were included.

Corrigendum 1 to ITU-T Rec. H.264 corrected and updated various minor aspects to bring the ITU-T version of the text up to date relative to the April 2005 output status approved as a new edition of the corresponding jointly-developed and technically-aligned text ISO/IEC 14496-10. It additionally fixed a number of minor errors and needs for clarification and defined three previously-reserved sample aspect ratio indicators.

Amendment 1 "Support of additional colour spaces and removal of the High 4:4:4 Profile" contained alterations to ITU-T Rec. H.264 | ISO/IEC 14496-10 Advanced Video Coding to specify the support of additional colour spaces and to remove the definition of the High 4:4:4 Profile.

NOTE – ITU-T Rec. H.264 is a twin text with ISO/IEC 14496-10 and this amendment was published in two different documents in the ISO/IEC series:

- The removal of the High 4:4:4 profile was found in ISO/IEC 14496-10:2005/Cor.2.
- The specification for support of additional colour spaces was found in ISO/IEC 14496-10:2005/Amd.1.

Amendment 2 "New profiles for professional applications" contained extensions to ITU-T Rec. H.264 | ISO/IEC 14496-10 Advanced Video Coding to specify the support of five additional profiles intended primarily for professional applications (the High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, CAVLC 4:4:4 Intra, and High 4:4:4 Predictive profiles) and two new types of supplemental enhancement information (SEI) messages (the post-filter hint SEI message and the tone mapping information SEI message).

Amendment 3 "Scalable video coding" contained extensions to ITU-T Rec. H.264 | ISO/IEC 14496-10 Advanced Video Coding to specify a scalable video coding extension in three profiles (the Scalable Baseline, Scalable High, and Scalable High Intra profiles).

The H.264 edition published in 2005-11 included the text approved 2005-03 and its Corrigendum 1 approved 2005-09. H.264 (2005) Amd.2 (2007) was available only as pre-published text since it was superseded by H.264 Amd.3 (2007-11) before its publication; further, H.264 Amd.3 was not published separately. This third edition integrated into the H.264 edition published in 2005-11 all changes approved in Amendments 1 (2006-06), 2 (2007-04) and 3 (2007-11).

Corrigendum 1 (2009) provides a significant number of minor corrections, clarifications, consistency improvements and formatting improvements drafted in response to accumulated errata reports collected since publication of the 2nd edition (dated 2005-03, which included a Cor.1 approved 2005-09).

The H.264 edition published in 2009-05 contained enhancement extensions to support multiview video coding (MVC), specification of a "Constrained Baseline Profile", and some miscellaneous corrections and clarifications.

This revision to ITU-T Rec. H.264 contains the specification of a new profile (the Stereo High profile) for two-view video coding with support of interlaced coding tools, the specification a new SEI message (the frame packing arrangement SEI message), and some miscellaneous corrections and clarifications.

History

Edition	Recommendation	Approval	Study Group
1.0	ITU-T H.264	2003-05-30	16
1.1	ITU-T H.264 (2003) Cor. 1	2004-05-07	16
2.0	ITU-T H.264	2005-03-01	16
2.1	ITU-T H.264 (2005) Cor. 1	2005-09-13	16
2.2	ITU-T H.264 (2005) Amend. 1	2006-06-13	16
2.3	ITU-T H.264 (2005) Amend. 2	2007-04-06	16
3.0	ITU-T H.264	2007-11-22	16
3.1	ITU-T H.264 (2007) Cor. 1	2009-01-13	16
4.0	ITU-T H.264	2009-03-16	16
5.0	ITU-T H.264	2010-03-09	16

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

3.27 **coded picture**: A *coded representation* of a *picture*. A coded picture may be either a *coded field* or a *coded frame*. Coded picture is a collective term referring to a *primary coded picture* or a *redundant coded picture*, but not to both together.

3.28 **coded picture buffer (CPB)**: A first-in first-out buffer containing *access units* in *decoding order* specified in the *hypothetical reference decoder* in Annex C.

3.29 **coded representation**: A data element as represented in its coded form.

3.30 **coded slice data partition NAL unit**: A *NAL unit* containing a *slice data partition*.

3.31 **coded slice NAL unit**: A *NAL unit* containing a *slice* that is not a *slice* of an *auxiliary coded picture*.

3.32 **coded video sequence**: A sequence of *access units* that consists, in decoding order, of an *IDR access unit* followed by zero or more non-IDR *access units* including all subsequent *access units* up to but not including any subsequent *IDR access unit*.

3.33 **component**: An array or single sample from one of the three arrays (*luma* and two *chroma*) that make up a *field* or *frame* in 4:2:0, 4:2:2, or 4:4:4 colour format or the array or a single sample of the array that make up a *field* or *frame* in monochrome format.

3.34 **complementary field pair:** A collective term for a *complementary reference field pair* or a *complementary non-reference field pair*.

3.35 **complementary non-reference field pair**: Two *non-reference fields* that are in consecutive *access units* in *decoding order* as two *coded fields* of opposite parity and share the same value of the frame_num *syntax element*, where the first *field* is not already a paired *field*.

3.36 **complementary reference field pair**: Two *reference fields* that are in consecutive *access units* in *decoding order* as two *coded fields* and share the same value of the frame_num *syntax element*, where the second *field* in *decoding order* is not an *IDR picture* and does not include a memory_management_control_operation *syntax element* equal to 5.

3.37 **context variable**: A variable specified for the *adaptive binary arithmetic decoding process* of a *bin* by an equation containing recently decoded *bins*.

3.38 **DC transform coefficient**: A *transform coefficient* for which the *frequency index* is zero in all dimensions.

3.39 **decoded picture**: A *decoded picture* is derived by decoding a *coded picture*. A *decoded picture* is either a decoded *frame*, or a decoded *field*. A decoded *field* is either a decoded *top field* or a decoded *bottom field*.

3.40 **decoded picture buffer (DPB)**: A buffer holding *decoded pictures* for reference, output reordering, or output delay specified for the *hypothetical reference decoder* in Annex C.

3.41 **decoder**: An embodiment of a *decoding process*.

3.42 **decoder under test (DUT)**: A *decoder* that is tested for conformance to this Recommendation | International Standard by operating the *hypothetical stream scheduler* to deliver a conforming *bitstream* to the *decoder* and to the *hypothetical reference decoder* and comparing the values and timing of the output of the two *decoders*.

3.43 **decoding order**: The order in which *syntax elements* are processed by the *decoding process*.

3.44 **decoding process**: The process specified in this Recommendation | International Standard that reads a *bitstream* and derives *decoded pictures* from it.

3.45 **direct prediction**: An *inter prediction* for a *block* for which no *motion vector* is decoded. Two direct *prediction* modes are specified that are referred to as spatial direct *prediction* and temporal *prediction* mode.

3.46 **display process**: A process not specified in this Recommendation | International Standard having, as its input, the cropped decoded *pictures* that are the output of the *decoding process*.

3.47 **emulation prevention byte**: A *byte* equal to 0x03 that may be present within a *NAL unit*. The presence of emulation prevention bytes ensures that no sequence of consecutive *byte-aligned bytes* in the *NAL unit* contains a *start code prefix*.

3.48 **encoder**: An embodiment of an *encoding process*.

3.49 **encoding process**: A process, not specified in this Recommendation | International Standard, that produces a *bitstream* conforming to this Recommendation | International Standard.

3.50 **field**: An assembly of alternate rows of a *frame*. A *frame* is composed of two *fields*, a *top field* and a *bottom field*.

3.51 **field macroblock**: A *macroblock* containing samples from a single *field*. All *macroblocks* of a *coded field* are field macroblocks. When *macroblock-adaptive frame/field decoding* is in use, some *macroblocks* of a *coded frame* may be field macroblocks.

3.52 **field macroblock pair**: A *macroblock pair* decoded as two *field macroblocks*.

3.53 **field scan**: A specific sequential ordering of *transform coefficients* that differs from the *zig-zag scan* by scanning columns more rapidly than rows. Field scan is used for *transform coefficients* in *field macroblocks*.

3.54 **flag**: A variable that can take one of the two possible values 0 and 1.

3.55 **frame**: A *frame* contains an array of *luma* samples in monochrome format or an array of *luma* samples and two corresponding arrays of *chroma* samples in 4:2:0, 4:2:2, and 4:4:4 colour format. A *frame* consists of two *fields*, a *top field* and a *bottom field*.

3.56 **frame macroblock**: A *macroblock* representing samples from the two *fields* of a *coded frame*. When *macroblock-adaptive frame/field decoding* is not in use, all *macroblocks* of a *coded frame* are frame macroblocks. When *macroblock-adaptive frame/field decoding* is in use, some *macroblocks* of a *coded frame* may be frame macroblocks.

3.57 **frame macroblock pair**: A *macroblock pair* decoded as two *frame macroblocks*.

3.58 **frequency index**: A one-dimensional or two-dimensional index associated with a *transform coefficient* prior to an *inverse transform* part of the *decoding process*.

3.59 **hypothetical reference decoder (HRD)**: A hypothetical *decoder* model that specifies constraints on the variability of conforming *NAL unit streams* or conforming *byte streams* that an encoding process may produce.

3.60 **hypothetical stream scheduler (HSS)**: A hypothetical delivery mechanism for the timing and data flow of the input of a *bitstream* into the *hypothetical reference decoder*. The HSS is used for checking the conformance of a *bitstream* or a *decoder*.

3.61 **I slice**: A *slice* that is not an *SI slice* that is decoded using *intra prediction* only.

3.62 **informative**: A term used to refer to content provided in this Recommendation | International Standard that is not an integral part of this Recommendation | International Standard. Informative content does not establish any mandatory requirements for conformance to this Recommendation | International Standard.

3.63 **instantaneous decoding refresh (IDR) access unit**: An *access unit* in which the *primary coded picture* is an *IDR picture*.

3.64 **instantaneous decoding refresh (IDR) picture**: A *coded picture* for which the variable IdrPicFlag is equal to 1. An IDR picture causes the *decoding process* to mark all *reference pictures* as "unused for reference" immediately after the decoding of the IDR picture. All *coded pictures* that follow an IDR picture in *decoding order* can be decoded without *inter prediction* from any *picture* that precedes the IDR picture in *decoding order*. The first *picture* of each *coded video sequence* in *decoding order* is an IDR picture.

3.65 **inter coding**: Coding of a *block*, *macroblock*, *slice*, or *picture* that uses *inter prediction*.

3.66 **inter prediction**: A *prediction* derived from decoded samples of *reference pictures* other than the current *decoded picture*.

3.67 **interpretation sample value**: A possibly-altered value corresponding to a decoded sample value of an *auxiliary coded picture* that may be generated for use in the *display process*. Interpretation sample values are not used in the *decoding process* and have no normative effect on the *decoding process*.

3.68 **intra coding**: Coding of a *block*, *macroblock*, *slice*, or *picture* that uses *intra prediction*.

3.69 **intra prediction**: A *prediction* derived from the decoded samples of the same decoded *slice*.

3.70 **intra slice**: See *I slice*.

3.71 **inverse transform**: A part of the *decoding process* by which a set of *transform coefficients* are converted into spatial-domain values, or by which a set of *transform coefficients* are converted into *DC transform coefficients*.

3.72 **layer**: One of a set of syntactical structures in a non-branching hierarchical relationship. Higher layers contain lower layers. The coding layers are the *coded video sequence*, *picture*, *slice*, and *macroblock* layers.

3.73 **level**: A defined set of constraints on the values that may be taken by the *syntax elements* and variables of this Recommendation | International Standard. The same set of levels is defined for all *profiles*, with most aspects of the definition of each level being in common across different *profiles*. Individual implementations may, within specified constraints, support a different level for each supported *profile*. In a different context, level is the value of a *transform coefficient* prior to *scaling*.

3.74 **list 0 (list 1) motion vector**: A *motion vector* associated with a *reference index* pointing into *reference picture list 0* (*list 1*).

3.75 **list 0 (list 1) prediction**: *Inter prediction* of the content of a *slice* using a *reference index* pointing into *reference picture list 0* (*list 1*).

3.76 **luma**: An adjective specifying that a sample array or single sample is representing the monochrome signal related to the primary colours. The symbol or subscript used for luma is Y or L.

NOTE – The term luma is used rather than the term luminance in order to avoid the implication of the use of linear light transfer characteristics that is often associated with the term luminance. The symbol L is sometimes used instead of the symbol Y to avoid confusion with the symbol y as used for vertical location.

3.77 **macroblock**: A 16x16 *block* of *luma* samples and two corresponding *blocks* of *chroma* samples of a *picture* that has three sample arrays, or a 16x16 *block* of samples of a monochrome *picture* or a *picture* that is coded using three separate colour planes. The division of a *slice* or a *macroblock pair* into macroblocks is a *partitioning*.

3.78 **macroblock-adaptive frame/field decoding**: A *decoding process* for *coded frames* in which some *macroblocks* may be decoded as *frame macroblocks* and others may be decoded as *field macroblocks*.

3.79 **macroblock address**: When *macroblock-adaptive frame/field decoding* is not in use, a macroblock address is the index of a *macroblock* in a *macroblock raster scan* of the *picture* starting with zero for the top-left *macroblock* in a *picture*. When *macroblock-adaptive frame/field decoding* is in use, the macroblock address of the *top macroblock* of a *macroblock pair* is two times the index of the *macroblock pair* in a *macroblock pair raster scan* of the *picture*, and the macroblock address of the *bottom macroblock* of a *macroblock pair* is the macroblock address of the corresponding *top macroblock* plus 1. The macroblock address of the *top macroblock* of each *macroblock pair* is an even number and the macroblock address of the *bottom macroblock* of each *macroblock pair* is an odd number.

3.80 **macroblock location**: The two-dimensional coordinates of a *macroblock* in a *picture* denoted by (x, y). For the top left *macroblock* of the *picture* (x, y) is equal to (0, 0). x is incremented by 1 for each *macroblock* column from left to right. When *macroblock-adaptive frame/field decoding* is not in use, y is incremented by 1 for each *macroblock* row from top to bottom. When *macroblock-adaptive frame/field decoding* is in use, y is incremented by 2 for each *macroblock pair* row from top to bottom, and is incremented by an additional 1 when a *macroblock* is a *bottom macroblock*.

3.81 **macroblock pair**: A pair of vertically contiguous *macroblocks* in a *frame* that is coupled for use in *macroblock-adaptive frame/field decoding*. The division of a *slice* into macroblock pairs is a *partitioning*.

3.82 **macroblock partition**: A *block* of *luma* samples and two corresponding *blocks* of *chroma* samples resulting from a *partitioning* of a *macroblock* for *inter prediction* for a *picture* that has three sample arrays or a *block* of *luma* samples resulting from a *partitioning* of a *macroblock* for *inter prediction* for a monochrome *picture* or a *picture* that is coded using three separate colour planes.

3.83 **macroblock to slice group map**: A means of mapping *macroblocks* of a *picture* into *slice groups*. The macroblock to slice group map consists of a list of numbers, one for each coded *macroblock*, specifying the *slice group* to which each coded *macroblock* belongs.

3.84 **map unit to slice group map**: A means of mapping *slice group map units* of a *picture* into *slice groups*. The map unit to slice group map consists of a list of numbers, one for each *slice group map unit*, specifying the *slice group* to which each coded *slice group map unit* belongs.

3.85 **may**: A term used to refer to behaviour that is allowed, but not necessarily required. In some places where the optional nature of the described behaviour is intended to be emphasized, the phrase "may or may not" is used to provide emphasis.

3.86 **memory management control operation**: Seven operations that control *reference picture marking*.

3.87 **motion vector**: A two-dimensional vector used for *inter prediction* that provides an offset from the coordinates in the *decoded picture* to the coordinates in a *reference picture*.

3.88 **must**: A term used in expressing an observation about a requirement or an implication of a requirement that is specified elsewhere in this Recommendation | International Standard. This term is used exclusively in an *informative* context.

3.89 **NAL unit**: A *syntax structure* containing an indication of the type of data to follow and *bytes* containing that data in the form of an *RBSP* interspersed as necessary with *emulation prevention bytes*.

3.90 **NAL unit stream**: A sequence of *NAL units*.

3.91 **non-paired field**: A collective term for a *non-paired reference field* or a *non-paired non-reference field*.

3.92 **non-paired non-reference field:** A decoded *non-reference field* that is not part of a *complementary non-reference field pair*.

3.93 **non-paired reference field**: A decoded *reference field* that is not part of a *complementary reference field pair*.

3.94 **non-reference field**: A *field* coded with nal_ref_idc equal to 0.

3.95 **non-reference frame**: A *frame* coded with nal_ref_idc equal to 0.

3.96 **non-reference picture**: A *picture* coded with nal_ref_idc equal to 0. A *non-reference picture* is not used for *inter prediction* of any other *pictures*.

3.97 **note**: A term used to prefix *informative* remarks. This term is used exclusively in an *informative* context.

3.98 **opposite parity**: The *opposite parity* of *top* is *bottom*, and vice versa.

3.99 **output order**: The order in which the *decoded pictures* are output from the *decoded picture buffer*.

3.100 **P slice**: A *slice* that is not an *SP slice* that may be decoded using *intra prediction* or *inter prediction* using at most one *motion vector* and *reference index* to *predict* the sample values of each *block*.

3.101 **parameter**: A *syntax element* of a *sequence parameter set* or a *picture parameter set*. Parameter is also used as part of the defined term *quantisation parameter*.

3.102 **parity**: The parity of a *field* can be *top* or *bottom*.

3.103 **partitioning**: The division of a set into subsets such that each element of the set is in exactly one of the subsets.

3.104 **picture**: A collective term for a *field* or a *frame*.

3.105 **picture parameter set**: A *syntax structure* containing *syntax elements* that apply to zero or more entire *coded pictures* as determined by the pic_parameter_set_id *syntax element* found in each *slice header*.

3.106 **picture order count**: A variable that is associated with each *coded field* and each *field* of a *coded frame* and has a value that is non-decreasing with increasing *field* position in *output order* relative to the first output *field* of the previous *IDR picture* in *decoding order* or relative to the first output *field* of the previous *picture*, in *decoding order*, that contains a *memory management control operation* that marks all *reference pictures* as "unused for reference".

3.107 **prediction**: An embodiment of the *prediction process*.

3.108 **prediction process**: The use of a *predictor* to provide an estimate of the sample value or data element currently being decoded.

3.109 **predictive slice**: See *P slice*.

3.110 **predictor**: A combination of specified values or previously decoded sample values or data elements used in the *decoding process* of subsequent sample values or data elements.

3.111 **primary coded picture**: The coded representation of a *picture* to be used by the *decoding process* for a bitstream conforming to this Recommendation | International Standard. The primary coded picture contains all *macroblocks* of the *picture*. The only *pictures* that have a normative effect on the *decoding process* are primary coded pictures. See also *redundant coded picture*.

3.112 **profile**: A specified subset of the syntax of this Recommendation | International Standard.

3.113 **quantisation parameter**: A variable used by the *decoding process* for *scaling* of *transform coefficient levels*.

3.114 **random access**: The act of starting the decoding process for a *bitstream* at a point other than the beginning of the stream.

3.115 **raster scan**: A mapping of a rectangular two-dimensional pattern to a one-dimensional pattern such that the first entries in the one-dimensional pattern are from the first top row of the two-dimensional pattern scanned

Exhibit Y

To

Joint Claim Chart

Overview of the H.264/AVC Video Coding Standard

Thomas Wiegand, Gary J. Sullivan, *Senior Member, IEEE*, Gisle Bjøntegaard, and Ajay Luthra, *Senior Member, IEEE*

***Abstract*—H.264/AVC is newest video coding standard of the ITU-T Video Coding Experts Group and the ISO/IEC Moving Picture Experts Group. The main goals of the H.264/AVC standardization effort have been enhanced compression performance and provision of a "network-friendly" video representation addressing "conversational" (video telephony) and "nonconversational" (storage, broadcast, or streaming) applications. H.264/AVC has achieved a significant improvement in rate-distortion efficiency relative to existing standards. This article provides an overview of the technical features of H.264/AVC, describes profiles and applications for the standard, and outlines the history of the standardization process.**

***Index Terms*—AVC, H.263, H.264, JVT, MPEG-2, MPEG-4, standards, video.**

I. Introduction

H.264/AVC is the newest international video coding standard [1]. By the time of this publication, it is expected to have been approved by ITU-T as Recommendation H.264 and by ISO/IEC as International Standard 14 496–10 (MPEG-4 part 10) Advanced Video Coding (AVC).

The MPEG-2 video coding standard (also known as ITU-T H.262) [2], which was developed about ten years ago primarily as an extension of prior MPEG-1 video capability with support of interlaced video coding, was an enabling technology for digital television systems worldwide. It is widely used for the transmission of standard definition (SD) and high definition (HD) TV signals over satellite, cable, and terrestrial emission and the storage of high-quality SD video signals onto DVDs.

However, an increasing number of services and growing popularity of high definition TV are creating greater needs for higher coding efficiency. Moreover, other transmission media such as Cable Modem, xDSL, or UMTS offer much lower data rates than broadcast channels, and enhanced coding efficiency can enable the transmission of more video channels or higher quality video representations within existing digital transmission capacities.

Video coding for telecommunication applications has evolved through the development of the ITU-T H.261, H.262 (MPEG-2), and H.263 video coding standards (and later enhancements of H.263 known as H.263+ and H.263 + +), and has diversified from ISDN and T1/E1 service to embrace PSTN, mobile wireless networks, and LAN/Internet network delivery. Throughout this evolution, continued efforts have been made to maximize coding efficiency while dealing with the diversification of network types and their characteristic formatting and loss/error robustness requirements.

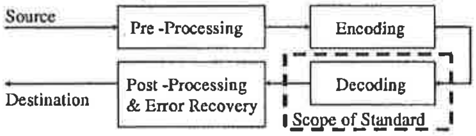


Fig. 1. Scope of video coding standardization.

Recently the MPEG-4 Visual (MPEG-4 part 2) standard [5] has also begun to emerge in use in some application domains of the prior coding standards. It has provided video shape coding capability, and has similarly worked toward broadening the range of environments for digital video use.

In early 1998, the *Video Coding Experts Group* (VCEG) ITU-T SG16 Q.6 issued a call for proposals on a project called H.26L, with the target to double the coding efficiency (which means halving the bit rate necessary for a given level of fidelity) in comparison to any other existing video coding standards for a broad variety of applications. The first draft design for that new standard was adopted in October of 1999. In December of 2001, VCEG and the *Moving Picture Experts Group* (MPEG) ISO/IEC JTC 1/SC 29/WG 11 formed a *Joint Video Team* (JVT), with the charter to finalize the draft new video coding standard for formal approval submission as H.264/AVC [1] in March 2003.

The scope of the standardization is illustrated in Fig. 1, which shows the typical video coding/decoding chain (excluding the transport or storage of the video signal). As has been the case for all ITU-T and ISO/IEC video coding standards, only the central decoder is standardized, by imposing restrictions on the bitstream and syntax, and defining the decoding process of the syntax elements such that every decoder conforming to the standard will produce similar output when given an encoded bitstream that conforms to the constraints of the standard. This limitation of the scope of the standard permits maximal freedom to optimize implementations in a manner appropriate to specific applications (balancing compression quality, implementation cost, time to market, etc.). However, it provides no guarantees of end-to-end reproduction quality, as it allows even crude encoding techniques to be considered conforming.

This paper is organized as follows. Section II provides a high-level overview of H.264/AVC applications and highlights some key technical features of the design that enable improved operation for this broad variety of applications. Section III explains the network abstraction layer (NAL) and the overall structure

Manuscript received April 15, 2002; revised May 10, 2003.

T. Wiegand is with the Fraunhofer-Institute for Telecommunications, Heinrich-Hertz-Institute, Einsteinufer 37, 10587 Berlin, Germany (e-mail: wiegand@hhi.de).

G. J. Sullivan is with the Microsoft Corporation, Redmond, WA 98052 USA (e-mail: garysull@microsoft.com).

G. Bjøntegaard is with the Tandberg, N-1324 Lysaker, Norway (e-mail: gbj@tandberg.no)

A. Luthra is with the Broadband Communications Sector, Motorola, Inc., San Diego, CA 92121 USA. (e-mail: aluthra@motorola.com)

Digital Object Identifier 10.1109/TCSVT.2003.815165

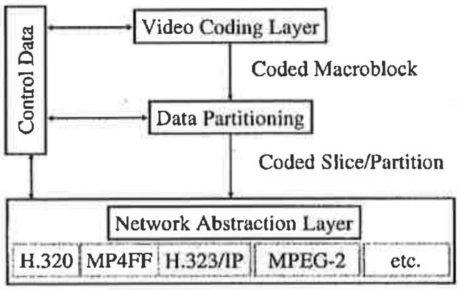


Fig. 2. Structure of H.264/AVC video encoder.

of H.264/AVC coded video data. The video coding layer (VCL) is described in Section IV. Section V explains the profiles supported by H.264/AVC and some potential application areas of the standard.

II. Applications and Design Feature Highlights

The new standard is designed for technical solutions including at least the following application areas

- Broadcast over cable, satellite, cable modem, DSL, terrestrial, etc.
- Interactive or serial storage on optical and magnetic devices, DVD, etc.
- Conversational services over ISDN, Ethernet, LAN, DSL, wireless and mobile networks, modems, etc. or mixtures of these.
- Video-on-demand or multimedia streaming services over ISDN, cable modem, DSL, LAN, wireless networks, etc.
- Multimedia messaging services (MMS) over ISDN, DSL, ethernet, LAN, wireless and mobile networks, etc.

Moreover, new applications may be deployed over existing and future networks. This raises the question about how to handle this variety of applications and networks.

To address this need for flexibility and customizability, the H.264/AVC design covers a VCL, which is designed to efficiently represent the video content, and a NAL, which formats the VCL representation of the video and provides header information in a manner appropriate for conveyance by a variety of transport layers or storage media (see Fig. 2).

Relative to prior video coding methods, as exemplified by MPEG-2 video, some highlighted features of the design that enable enhanced coding efficiency include the following enhancements of the ability to predict the values of the content of a picture to be encoded.

- **Variable block-size motion compensation with small block sizes:** This standard supports more flexibility in the selection of motion compensation block sizes and shapes than any previous standard, with a minimum luma motion compensation block size as small as 4×4.
- **Quarter-sample-accurate motion compensation:** Most prior standards enable half-sample motion vector accuracy at most. The new design improves up on this by adding quarter-sample motion vector accuracy, as first found in an advanced profile of the MPEG-4 Visual (part 2) standard, but further reduces the complexity of the interpolation processing compared to the prior design.
- **Motion vectors over picture boundaries:** While motion vectors in MPEG-2 and its predecessors were required to point only to areas within the previously-decoded reference picture, the picture boundary extrapolation technique first found as an optional feature in H.263 is included in H.264/AVC.
- **Multiple reference picture motion compensation:** Predictively coded pictures (called "P" pictures) in MPEG-2 and its predecessors used only one previous picture to predict the values in an incoming picture. The new design extends upon the enhanced reference picture selection technique found in H.263++ to enable efficient coding by allowing an encoder to select, for motion compensation purposes, among a larger number of pictures that have been decoded and stored in the decoder. The same extension of referencing capability is also applied to motion-compensated bi-prediction, which is restricted in MPEG-2 to using two specific pictures only (one of these being the previous intra (I) or P picture in display order and the other being the next I or P picture in display order).
- **Decoupling of referencing order from display order:** In prior standards, there was a strict dependency between the ordering of pictures for motion compensation referencing purposes and the ordering of pictures for display purposes. In H.264/AVC, these restrictions are largely removed, allowing the encoder to choose the ordering of pictures for referencing and display purposes with a high degree of flexibility constrained only by a total memory capacity bound imposed to ensure decoding ability. Removal of the restriction also enables removing the extra delay previously associated with bi-predictive coding.
- **Decoupling of picture representation methods from picture referencing capability:** In prior standards, pictures encoded using some encoding methods (namely bi-predictively-encoded pictures) could not be used as references for prediction of other pictures in the video sequence. By removing this restriction, the new standard provides the encoder more flexibility and, in many cases, an ability to use a picture for referencing that is a closer approximation to the picture being encoded.
- **Weighted prediction:** A new innovation in H.264/AVC allows the motion-compensated prediction signal to be weighted and offset by amounts specified by the encoder. This can dramatically improve coding efficiency for scenes containing fades, and can be used flexibly for other purposes as well.
- **Improved "skipped" and "direct" motion inference:** In prior standards, a "skipped" area of a predictively-coded picture could not motion in the scene content. This had a detrimental effect when coding video containing global motion, so the new H.264/AVC design instead infers motion in "skipped" areas. For bi-predictively coded areas (called B slices), H.264/AVC also includes an enhanced motion inference method known as "direct" motion compensation, which improves further on prior "direct" prediction designs found in H.263+ and MPEG-4 Visual.

- **Directional spatial prediction for intra coding:** A new technique of extrapolating the edges of the previously-decoded parts of the current picture is applied in regions of pictures that are coded as intra (i.e., coded without reference to the content of some other picture). This improves the quality of the prediction signal, and also allows prediction from neighboring areas that were not coded using intra coding (something not enabled when using the transform-domain prediction method found in H.263+ and MPEG-4 Visual).
- **In-the-loop deblocking filtering:** Block-based video coding produces artifacts known as blocking artifacts. These can originate from both the prediction and residual difference coding stages of the decoding process. Application of an adaptive deblocking filter is a well-known method of improving the resulting video quality, and when designed well, this can improve both objective and subjective video quality. Building further on a concept from an optional feature of H.263+, the deblocking filter in the H.264/AVC design is brought within the motion-compensated prediction loop, so that this improvement in quality can be used in inter-picture prediction to improve the ability to predict other pictures as well.

In addition to improved prediction methods, other parts of the design were also enhanced for improved coding efficiency, including the following.

- **Small block-size transform:** All major prior video coding standards used a transform block size of 8×8, while the new H.264/AVC design is based primarily on a 4×4 transform. This allows the encoder to represent signals in a more locally-adaptive fashion, which reduces artifacts known colloquially as "ringing". (The smaller block size is also justified partly by the advances in the ability to better predict the content of the video using the techniques noted above, and by the need to provide transform regions with boundaries that correspond to those of the smallest prediction regions.)
- **Hierarchical block transform:** While in most cases, using the small 4×4 transform block size is perceptually beneficial, there are some signals that contain sufficient correlation to call for some method of using a representation with longer basis functions. The H.264/AVC standard enables this in two ways: 1) by using a hierarchical transform to extend the effective block size use for low-frequency chroma information to an 8×8 array and 2) by allowing the encoder to select a special coding type for intra coding, enabling extension of the length of the luma transform for low-frequency information to a 16×16 block size in a manner very similar to that applied to the chroma.
- **Short word-length transform:** All prior standard designs have effectively required encoders and decoders to use more complex processing for transform computation. While previous designs have generally required 32-bit processing, the H.264/AVC design requires only 16-bit arithmetic.
- **Exact-match inverse transform:** In previous video coding standards, the transform used for representing the video was generally specified only within an error tolerance bound, due to the impracticality of obtaining an exact match to the ideal specified inverse transform. As a result, each decoder design would produce slightly different decoded video, causing a "drift" between encoder and decoder representation of the video and reducing effective video quality. Building on a path laid out as an optional feature in the H.263 + + effort, H.264/AVC is the first standard to achieve exact equality of decoded video content from all decoders.
- **Arithmetic entropy coding:** An advanced entropy coding method known as arithmetic coding is included in H.264/AVC. While arithmetic coding was previously found as an optional feature of H.263, a more effective use of this technique is found in H.264/AVC to create a very powerful entropy coding method known as CABAC (context-adaptive binary arithmetic coding).
- **Context-adaptive entropy coding:** The two entropy coding methods applied in H.264/AVC, termed CAVLC (context-adaptive variable-length coding) and CABAC, both use context-based adaptivity to improve performance relative to prior standard designs.

Robustness to data errors/losses and flexibility for operation over a variety of network environments is enabled by a number of design aspects new to the H.264/AVC standard, including the following highlighted features.

- **Parameter set structure:** The parameter set design provides for robust and efficient conveyance header information. As the loss of a few key bits of information (such as sequence header or picture header information) could have a severe negative impact on the decoding process when using prior standards, this key information was separated for handling in a more flexible and specialized manner in the H.264/AVC design.
- **NAL unit syntax structure:** Each syntax structure in H.264/AVC is placed into a logical data packet called a NAL unit. Rather than forcing a specific bitstream interface to the system as in prior video coding standards, the NAL unit syntax structure allows greater customization of the method of carrying the video content in a manner appropriate for each specific network.
- **Flexible slice size:** Unlike the rigid slice structure found in MPEG-2 (which reduces coding efficiency by increasing the quantity of header data and decreasing the effectiveness of prediction), slice sizes in H.264/AVC are highly flexible, as was the case earlier in MPEG-1.
- **Flexible macroblock ordering (FMO):** A new ability to partition the picture into regions called slice groups has been developed, with each slice becoming an independently-decodable subset of a slice group. When used effectively, flexible macroblock ordering can significantly enhance robustness to data losses by managing the spatial relationship between the regions that are coded in each slice. (FMO can also be used for a variety of other purposes as well.)
- **Arbitrary slice ordering (ASO):** Since each slice of a coded picture can be (approximately) decoded independently of the other slices of the picture, the H.264/AVC

design enables sending and receiving the slices of the picture in any order relative to each other. This capability, first found in an optional part of H.263+, can improve end-to-end delay in real-time applications, particularly when used on networks having out-of-order delivery behavior (e.g., internet protocol networks).

- **Redundant pictures:** In order to enhance robustness to data loss, the H.264/AVC design contains a new ability to allow an encoder to send redundant representations of regions of pictures, enabling a (typically somewhat degraded) representation of regions of pictures for which the primary representation has been lost during data transmission.
- **Data Partitioning:** Since some coded information for representation of each region (e.g., motion vectors and other prediction information) is more important or more valuable than other information for purposes of representing the video content, H.264/AVC allows the syntax of each slice to be separated into up to three different partitions for transmission, depending on a categorization of syntax elements. This part of the design builds further on a path taken in MPEG-4 Visual and in an optional part of H.263 + +. Here, the design is simplified by having a single syntax with partitioning of that same syntax controlled by a specified categorization of syntax elements.
- **SP/SI synchronization/switching pictures:** The H.264/AVC design includes a new feature consisting of picture types that allow exact synchronization of the decoding process of some decoders with an ongoing video stream produced by other decoders without penalizing all decoders with the loss of efficiency resulting from sending an I picture. This can enable switching a decoder between representations of the video content that used different data rates, recovery from data losses or errors, as well as enabling trick modes such as fast-forward, fast-reverse, etc.

In Sections III and IV, a more detailed description of the key features is given.

III. NAL

The NAL is designed in order to provide "network friendliness" to enable simple and effective customization of the use of the VCL for a broad variety of systems.

The NAL facilitates the ability to map H.264/AVC VCL data to transport layers such as:

- RTP/IP for any kind of real-time wire-line and wireless Internet services (conversational and streaming);
- File formats, e.g., ISO MP4 for storage and MMS;
- H.32X for wireline and wireless conversational services;
- MPEG-2 systems for broadcasting services, etc.

The full degree of customization of the video content to fit the needs of each particular application is outside the scope of the H.264/AVC standardization effort, but the design of the NAL anticipates a variety of such mappings. Some key concepts of the NAL are NAL units, byte stream, and packet format uses of NAL units, parameter sets, and access units. A short description of these concepts is given below whereas a more detailed description including error resilience aspects is provided in [6] and [7].

A. NAL Units

The coded video data is organized into NAL units, each of which is effectively a packet that contains an integer number of bytes. The first byte of each NAL unit is a header byte that contains an indication of the type of data in the NAL unit, and the remaining bytes contain payload data of the type indicated by the header.

The payload data in the NAL unit is interleaved as necessary with *emulation prevention bytes*, which are bytes inserted with a specific value to prevent a particular pattern of data called a *start code prefix* from being accidentally generated inside the payload.

The NAL unit structure definition specifies a generic format for use in both packet-oriented and bitstream-oriented transport systems, and a series of NAL units generated by an encoder is referred to as a NAL unit stream.

B. NAL Units in Byte-Stream Format Use

Some systems (e.g., H.320 and MPEG-2/H.222.0 systems) require delivery of the entire or partial NAL unit stream as an ordered stream of bytes or bits within which the locations of NAL unit boundaries need to be identifiable from patterns within the coded data itself.

For use in such systems, the H.264/AVC specification defines a byte stream format. In the byte stream format, each NAL unit is prefixed by a specific pattern of three bytes called a start code prefix. The boundaries of the NAL unit can then be identified by searching the coded data for the unique start code prefix pattern. The use of emulation prevention bytes guarantees that start code prefixes are unique identifiers of the start of a new NAL unit.

A small amount of additional data (one byte per video picture) is also added to allow decoders that operate in systems that provide streams of bits without alignment to byte boundaries to recover the necessary alignment from the data in the stream.

Additional data can also be inserted in the byte stream format that allows expansion of the amount of data to be sent and can aid in achieving more rapid byte alignment recovery, if desired.

C. NAL Units in Packet-Transport System Use

In other systems (e.g., internet protocol/RTP systems), the coded data is carried in packets that are framed by the system transport protocol, and identification of the boundaries of NAL units within the packets can be established without use of start code prefix patterns. In such systems, the inclusion of start code prefixes in the data would be a waste of data carrying capacity, so instead the NAL units can be carried in data packets without start code prefixes.

D. VCL and Non-VCL NAL Units

NAL units are classified into VCL and non-VCL NAL units. The VCL NAL units contain the data that represents the values of the samples in the video pictures, and the non-VCL NAL units contain any associated additional information such as parameter sets (important header data that can apply to a large

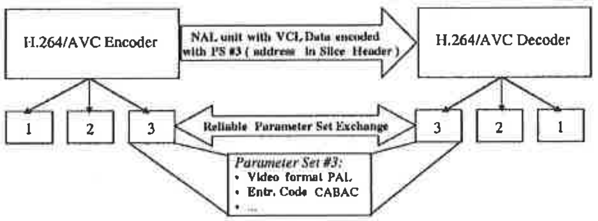

H.264/AVC Encoder
NAL unit with VCL Data encoded with PS #3 (address in Slice Header)
H.264/AVC Decoder
1
2
3
Reliable Parameter Set Exchange
3
2
1
Parameter Set #3:
• Video format PAL
• Entr. Code CABAC
• ...

Fig. 3. Parameter set use with reliable "out-of-band" parameter set exchange.

number of VCL NAL units) and supplemental enhancement information (timing information and other supplemental data that may enhance usability of the decoded video signal but are not necessary for decoding the values of the samples in the video pictures).

E. Parameter Sets

A parameter set is supposed to contain information that is expected to rarely change and offers the decoding of a large number of VCL NAL units. There are two types of parameter sets:

- sequence parameter sets, which apply to a series of consecutive coded video pictures called a coded video sequence;
- picture parameter sets, which apply to the decoding of one or more individual pictures within a coded video sequence.

The sequence and picture parameter-set mechanism decouples the transmission of infrequently changing information from the transmission of coded representations of the values of the samples in the video pictures. Each VCL NAL unit contains an identifier that refers to the content of the relevant picture parameter set and each picture parameter set contains an identifier that refers to the content of the relevant sequence parameter set. In this manner, a small amount of data (the identifier) can be used to refer to a larger amount of information (the parameter set) without repeating that information within each VCL NAL unit.

Sequence and picture parameter sets can be sent well ahead of the VCL NAL units that they apply to, and can be repeated to provide robustness against data loss. In some applications, parameter sets may be sent within the channel that carries the VCL NAL units (termed "in-band" transmission). In other applications (see Fig. 3), it can be advantageous to convey the parameter sets "out-of-band" using a more reliable transport mechanism than the video channel itself.

F. Access Units

A set of NAL units in a specified form is referred to as an access unit. The decoding of each access unit results in one decoded picture. The format of an access unit is shown in Fig. 4.

Each access unit contains a set of VCL NAL units that together compose a *primary coded picture*. It may also be prefixed with an *access unit delimiter* to aid in locating the start of the access unit. Some *supplemental enhancement information* containing data such as picture timing information may also precede the primary coded picture.

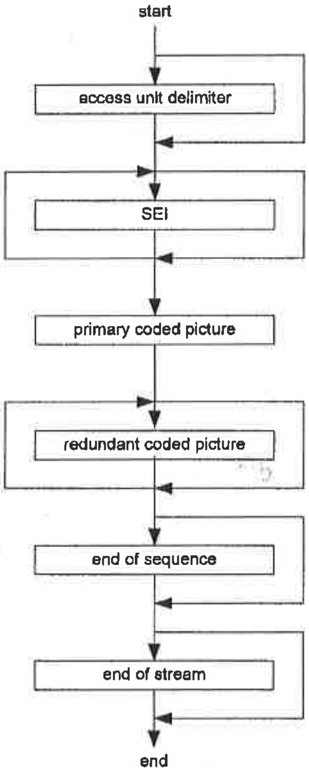


Fig. 4. Structure of an access unit.

The primary coded picture consists of a set of VCL NAL units consisting of *slices* or *slice data partitions* that represent the samples of the video picture.

Following the primary coded picture may be some additional VCL NAL units that contain redundant representations of areas of the same video picture. These are referred to as *redundant coded pictures*, and are available for use by a decoder in recovering from loss or corruption of the data in the primary coded pictures. Decoders are not required to decode redundant coded pictures if they are present.

Finally, if the coded picture is the last picture of a coded video sequence (a sequence of pictures that is independently decodable and uses only one sequence parameter set), an *end of sequence* NAL unit may be present to indicate the end of the sequence; and if the coded picture is the last coded picture in the entire NAL unit stream, an *end of stream* NAL unit may be present to indicate that the stream is ending.

G. Coded Video Sequences

A coded video sequence consists of a series of access units that are sequential in the NAL unit stream and use only one sequence parameter set. Each coded video sequence can be decoded independently of any other coded video sequence, given the necessary parameter set information, which may be conveyed "in-band" or "out-of-band". At the beginning of a coded video sequence is an *instantaneous decoding refresh* (IDR) access unit. An IDR access unit contains an *intra* picture—a coded

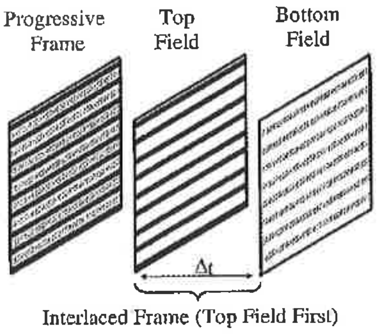


Fig. 5. Progressive and interlaced frames and fields.

picture that can be decoded without decoding any previous pictures in the NAL unit stream, and the presence of an IDR access unit indicates that no subsequent picture in the stream will require reference to pictures prior to the intra picture it contains in order to be decoded.

A NAL unit stream may contain one or more coded video sequences.

IV. VCL

As in all prior ITU-T and ISO/IEC JTC1 video standards since H.261 [3], the VCL design follows the so-called block-based hybrid video coding approach (as depicted in Fig. 8), in which each coded picture is represented in block-shaped units of associated luma and chroma samples called *macroblocks*. The basic source-coding algorithm is a hybrid of inter-picture prediction to exploit temporal statistical dependencies and transform coding of the prediction residual to exploit spatial statistical dependencies. There is no single coding element in the VCL that provides the majority of the significant improvement in compression efficiency in relation to prior video coding standards. It is rather a plurality of smaller improvements that add up to the significant gain.

A. Pictures, Frames, and Fields

A coded video sequence in H.264/AVC consists of a sequence of *coded pictures*. A coded picture in [1] can represent either an entire *frame* or a single *field*, as was also the case for MPEG-2 video.

Generally, a frame of video can be considered to contain two interleaved fields, a top and a bottom field. The top field contains even-numbered rows 0, 2,...,H/2–1 with H being the number of rows of the frame. The bottom field contains the odd-numbered rows (starting with the second line of the frame). If the two fields of a frame were captured at different time instants, the frame is referred to as an interlaced frame, and otherwise it is referred to as a progressive frame (see Fig. 5). The coding representation in H.264/AVC is primarily agnostic with respect to this video characteristic, i.e., the underlying interlaced or progressive timing of the original captured pictures. Instead, its coding specifies a representation based primarily on geometric concepts rather than being based on timing.

B. YCbCr Color Space and 4:2:0 Sampling

The human visual system seems to perceive scene content in terms of brightness and color information separately, and with greater sensitivity to the details of brightness than color. Video transmission systems can be designed to take advantage of this. (This is true of conventional analog TV systems as well as digital ones.) In H.264/AVC as in prior standards, this is done by using a YCbCr color space together with reducing the sampling resolution of the Cb and Cr chroma information.

The video color space used by H.264/AVC separates a color representation into three components called Y, Cb, and Cr. Component Y is called *luma*, and represents brightness. The two *chroma* components Cb and Cr represent the extent to which the color deviates from gray toward blue and red, respectively. (The terms luma and chroma are used in this paper and in the standard rather than the terms luminance and chrominance, in order to avoid the implication of the use of linear light transfer characteristics that is often associated with the terms luminance and chrominance.)

Because the human visual system is more sensitive to luma than chroma, H.264/AVC uses a sampling structure in which the chroma component has one fourth of the number of samples than the luma component (half the number of samples in both the horizontal and vertical dimensions). This is called 4:2:0 sampling with 8 bits of precision per sample. The sampling structure used is the same as in MPEG-2 Main-profile video. (Proposals for extension of the standard to also support higher-resolution chroma and a larger number of bits per sample are currently being considered.)

C. Division of the Picture Into Macroblocks

A picture is partitioned into fixed-size macroblocks that each cover a rectangular picture area of 16×16 samples of the luma component and 8×8 samples of each of the two chroma components. This partitioning into macroblocks has been adopted into all previous ITU-T and ISO/IEC JTC1 video coding standards since H.261 [3]. Macroblocks are the basic building blocks of the standard for which the decoding process is specified. The basic coding algorithm for a macroblock is described after we explain how macroblocks are grouped into slices.

D. Slices and Slice Groups

Slices are a sequence of macroblocks which are processed in the order of a raster scan when not using FMO which is described in the next paragraph. A picture maybe split into one or several slices as shown in Fig. 6. A picture is therefore a collection of one or more slices in H.264/AVC. Slices are self-contained in the sense that given the active sequence and picture parameter sets, their syntax elements can be parsed from the bitstream and the values of the samples in the area of the picture that the slice represents can be correctly decoded without use of data from other slices provided that utilized reference pictures are identical at encoder and decoder. Some information from other slices maybe needed to apply the deblocking filter across slice boundaries.

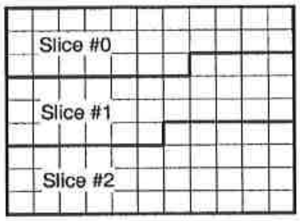


Fig. 6. Subdivision of a picture into slices when not using FMO.

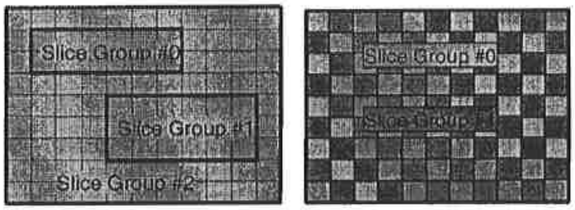


Fig. 7. Subdivision of a QCIF frame into slices when utilizing FMO.

FMO modifies the way how pictures are partitioned into slices and macroblocks by utilizing the concept of *slice groups*. Each slice group is a set of macroblocks defined by a *macroblock to slice group map*, which is specified by the content of the picture parameter set and some information from slice headers. The macroblock to slice group map consists of a slice group identification number for each macroblock in the picture, specifying which slice group the associated macroblock belongs to. Each slice group can be partitioned into one or more slices, such that a slice is a sequence of macroblocks within the same slice group that is processed in the order of a raster scan within the set of macroblocks of a particular slice group. (The case when FMO is not in use can be viewed as the simple special case of FMO in which the whole picture consists of a single slice group.)

Using FMO, a picture can be split into many macroblock scanning patterns such as interleaved slices, a dispersed macroblock allocation, one or more "foreground" slice groups and a "leftover" slice group, or a checker-board type of mapping. The latter two are illustrated in Fig. 7. The left-hand side macroblock to slice group mapping has been demonstrated for use in region-of-interest type of coding applications. The right-hand side macroblock to slice group mapping has been demonstrated useful for concealment in video conferencing applications where slice group #0 and slice group #1 are transmitted in separate packets and one of them is lost. For more details on the use of FMO, see [14].

Regardless of whether FMO is in use or not, each slice can be coded using different coding types as follows.

- **I slice:** A slice in which all macroblocks of the slice are coded using intra prediction.
- **P slice:** In addition to the coding types of the I slice, some macroblocks of the P slice can also be coded using inter prediction with at most *one* motion-compensated prediction signal per prediction block.
- **B slice:** In addition to the coding types available in a P slice, some macroblocks of the B slice can also be coded using inter prediction with *two* motion-compensated prediction signals per prediction block.

The above three coding types are very similar to those in previous standards with the exception of the use of reference pictures as described below. The following two coding types for slices are new.

- **SP slice:** A so-called switching P slice that is coded such that efficient switching between different pre-coded pictures becomes possible.
- **SI slice:** A so-called switching I slice that allows an exact match of a macroblock in an SP slice for random access and error recovery purposes.

For details on the novel concept of SP and SI slices, the reader is referred to [5], while the other slice types are further described below.

E. Encoding and Decoding Process for Macroblocks

All luma and chroma samples of a macroblock are either spatially or temporally predicted, and the resulting prediction residual is encoded using transform coding. For transform coding purposes, each color component of the prediction residual signal is subdivided into smaller 4×4 blocks. Each block is transformed using an integer transform, and the transform coefficients are quantized and encoded using entropy coding methods.

Fig. 8 shows a block diagram of the VCL for a macroblock. The input video signal is split into macroblocks, the association of macroblocks to slice groups and slices is selected, and then each macroblock of each slice is processed as shown. An efficient parallel processing of macroblocks is possible when there are various slices in the picture.

F. Adaptive Frame/Field Coding Operation

In interlaced frames with regions of moving objects or camera motion, two adjacent rows tend to show a reduced degree of statistical dependency when compared to progressive frames in. In this case, it may be more efficient to compress each field separately. To provide high coding efficiency, the H.264/AVC design allows encoders to make any of the following decisions when coding a frame.

1) To combine the two fields together and to code them as one single coded frame (frame mode).
2) To not combine the two fields and to code them as separate coded fields (field mode).
3) To combine the two fields together and compress them as a single frame, but when coding the frame to split the pairs of two vertically adjacent macroblocks into either pairs of two field or frame macroblocks before coding them.

The choice between the three options can be made adaptively for each frame in a sequence. The choice between the first two options is referred to as picture-adaptive frame/field (PAFF) coding. When a frame is coded as two fields, each field is partitioned into macroblocks and is coded in a manner very similar to a frame, with the following main exceptions:

- motion compensation utilizes reference fields rather than reference frames;
- the zig-zag scan of transform coefficients is different;

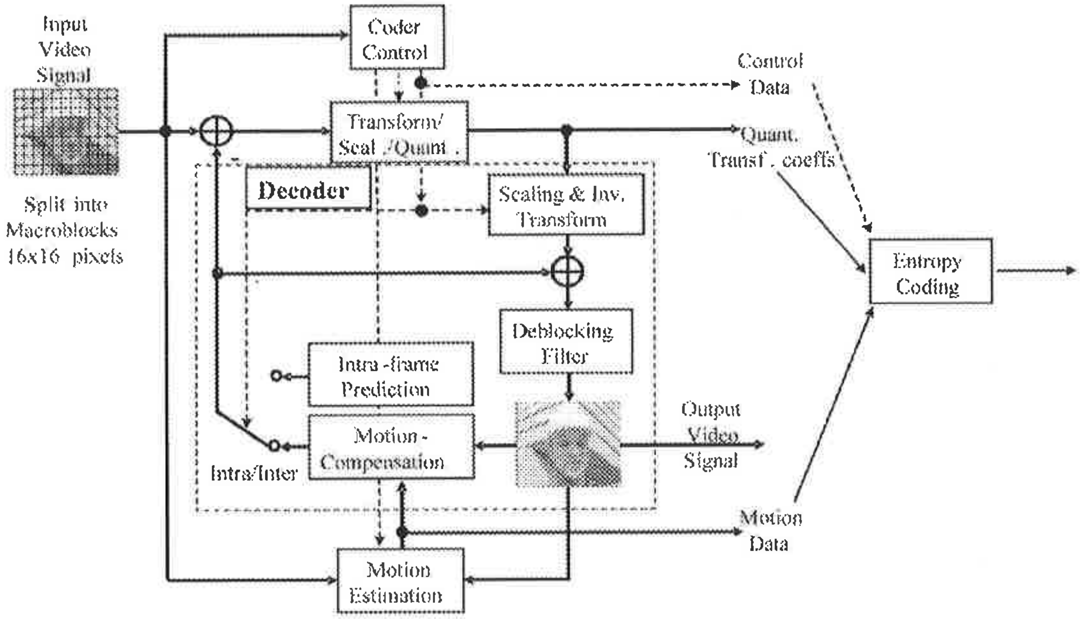


Fig. 8. Basic coding structure for H.264/AVC for a macroblock.

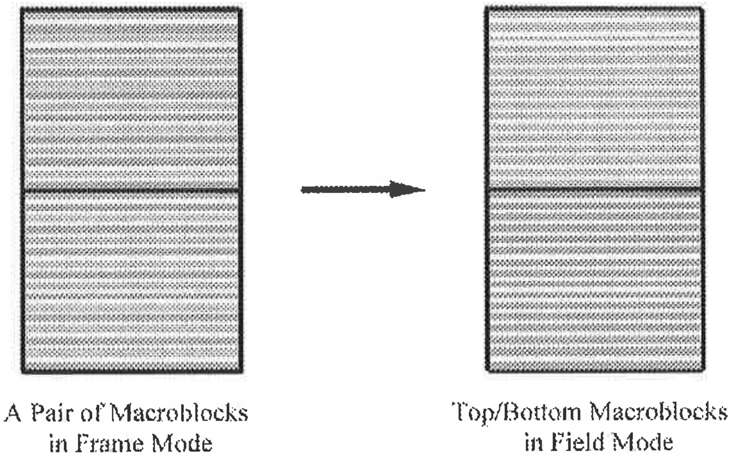


Fig. 9. Conversion of a frame macroblock pair into a field macroblock pair.

- the strong deblocking strength is not used for filtering horizontal edges of macroblocks in fields, because the field rows are spatially twice as far apart as frame rows and the length of the filter thus covers a larger spatial area.

During the development of the H.264/AVC standard, PAFF coding was reported to reduce bit rates in the range of 16% to 20% over frame-only coding mode for ITU-R 601 resolution sequences like "Canoa", "Rugby", etc.

If a frame consists of mixed regions where some regions are moving and others are not, it is typically more efficient to code the nonmoving regions in frame mode and the moving regions in the field mode. Therefore, the frame/field encoding decision can also be made independently for each vertical pair of macroblocks (a 16×32 luma region) in a frame. This coding option is referred to as macroblock-adaptive frame/field (MBAFF) coding. For a macroblock pair that is coded in frame mode, each macroblock contains frame lines. For a macroblock pair that is coded in field mode, the top macroblock contains top field lines and the bottom macroblock contains bottom field lines. Fig. 9 illustrates the MBAFF macroblock pair concept.

Note that, unlike in MPEG-2, the frame/field decision is made at the macroblock pair level rather than within the macroblock level. The reasons for this choice are to keep the basic macroblock processing structure intact, and to permit motion compensation areas as large as the size of a macroblock.

Each macroblock of a field macroblock pair is processed very similarly to a macroblock within a field in PAFF coding. However, since a mixture of field and frame macroblock pairs may occur within an MBAFF frame, the methods that are used for zig-zag scanning, prediction of motion vectors, prediction of intra prediction modes, intra-frame sample prediction, deblocking filtering, and context modeling in entropy coding are modified to account for this mixture. The main idea is to preserve as much spatial consistency as possible. It should be noted that the specification of spatial neighbors in MBAFF frames is rather complicated (please refer to [1]) and that in Sections IV-G–L spatial neighbors are only described for non-MBAFF frames.

Another important distinction between MBAFF and PAFF is that in MBAFF, one field cannot use the macroblocks in the other field of the same frame as a reference for motion prediction (because some regions of each field are not yet available when a field macroblock of the other field is coded). Thus, sometimes PAFF coding can be more efficient than MBAFF coding (particularly in the case of rapid global motion, scene change, or intra picture refresh), although the reverse is usually true.

During the development of the standard, MBAFF was reported to reduce bit rates in the range of 14 to 16% over PAFF for ITU-R 601 resolution sequences like "Mobile and Calendar" and "MPEG-4 World News".

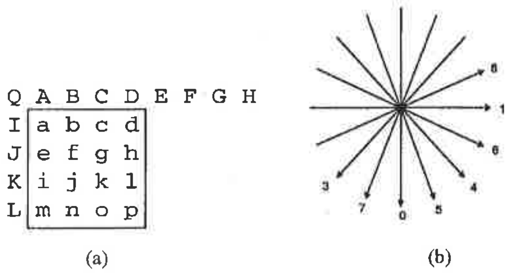
G. Intra-Frame Prediction

Each macroblock can be transmitted in one of several coding types depending on the slice-coding type. In all slice-coding types, the following types of intra coding are supported, which are denoted as Intra_4×4 or Intra_16×16 together with chroma prediction and I_PCM prediction modes.

The Intra_4×4 mode is based on predicting each 4×4 luma block separately and is well suited for coding of parts of a picture with significant detail. The Intra_16×16 mode, on the other hand, performs prediction of the whole 16×16 luma block and is more suited for coding very smooth areas of a picture. In addition to these two types of luma prediction, a separate chroma prediction is conducted. As an alternative to Intra_4×4 and Intra_16×16, the I_PCM coding type allows the encoder to simply bypass the prediction and transform coding processes and instead directly send the values of the encoded samples. The I_PCM mode serves the following purposes.

1) It allows the encoder to precisely represent the values of the samples.
2) It provides a way to accurately represent the values of anomalous picture content without significant data expansion
3) It enables placing a hard limit on the number of bits a decoder must handle for a macroblock without harm to coding efficiency

In contrast to some previous video coding standards (namely H.263+ and MPEG-4 Visual), where intra prediction has been conducted in the transform domain intra prediction in H.264/AVC is always conducted in the spatial domain, by referring to neighboring samples of previously-coded blocks which are to the left and/or above the block to be predicted. This may incur error propagation in environments with transmission errors that propagate due to motion compensation into inter-coded macroblocks. Therefore, a constrained intra coding mode can be signaled that allows prediction only from intra-coded neighboring macroblocks.



(a) (b)

Fig. 10. (a) Intra_4×4 prediction is conducted for samples a-p of a block using samples A-Q. (b) Eight "prediction directions" for Intra_4×4 prediction.

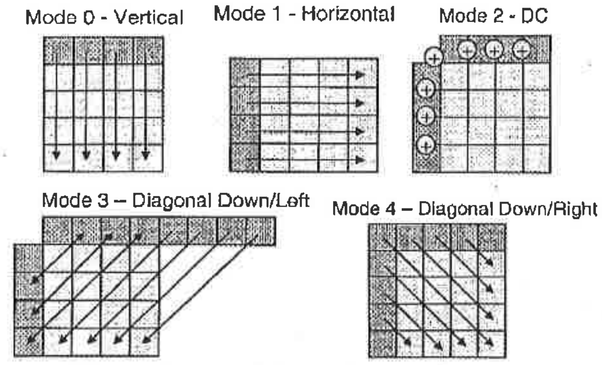


Fig. 11. Five of the nine Intra_4×4 prediction modes.

When using the Intra_4×4 mode, each 4×4 block is predicted from spatially neighboring samples as illustrated on the left-hand side of Fig. 10. The 16 samples of the 4×4 block which are labeled as a-p are predicted using prior decoded samples in adjacent blocks labeled as A-Q. For each 4×4 block, one of nine prediction modes can be utilized. In addition to "DC" prediction (where one value is used to predict the entire 4×4 block), eight directional prediction modes are specified as illustrated on the right-hand side of Fig. 10. Those modes are suitable to predict directional structures in a picture such as edges at various angles.

Fig. 11 shows five of the nine Intra_4×4 prediction modes. For mode 0 (vertical prediction), the samples above the 4×4 block are copied into the block as indicated by the arrows. Mode 1 (horizontal prediction) operates in a manner similar to vertical prediction except that the samples to the left of the 4×4 block are copied. For mode 2 (DC prediction), the adjacent samples are averaged as indicated in Fig. 11. The remaining six modes are diagonal prediction modes which are called diagonal-down-left, diagonal-down-right, vertical-right, horizontal-down, vertical-left, and horizontal-up prediction. As their names indicate, they are suited to predict textures with structures in the specified direction. The first two diagonal prediction modes are also illustrated in Fig. 11. When samples E-H (Fig. 10) that are used for the diagonal-down-left prediction mode are not available (because they have not yet been decoded or they are outside of the slice or not in an intra-coded macroblock in the constrained intra-mode), these samples are replaced by sample D. Note that in earlier draft versions of the Intra_4×4 prediction mode the four samples below sample L were also used for some prediction modes. However, due to the need to reduce memory access,

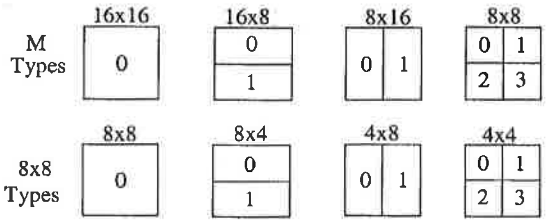


Fig. 12. Segmentations of the macroblock for motion compensation. Top: segmentation of macroblocks, bottom: segmentation of 8×8 partitions.

these have been dropped, as the relative gain for their use is very small.

When utilizing the Intra_16×16 mode, the whole luma component of a macroblock is predicted. Four prediction modes are supported. Prediction mode 0 (vertical prediction), mode 1 (horizontal prediction), and mode 2 (DC prediction) are specified similar to the modes in Intra_4×4 prediction except that instead of 4 neighbors on each side to predict a 4×4 block, 16 neighbors on each side to predict a 16×16 block are used. For the specification of prediction mode 4 (plane prediction), please refer to [1].

The chroma samples of a macroblock are predicted using a similar prediction technique as for the luma component in Intra_16×16 macroblocks, since chroma is usually smooth over large areas.

Intra prediction (and all other forms of prediction) across slice boundaries is not used, in order to keep all slices independent of each other.

H. Inter-Frame Prediction

1) Inter-Frame Prediction in P Slices: In addition to the intra macroblock coding types, various *predictive* or motion-compensated coding types are specified as P macroblock types. Each P macroblock type corresponds to a specific partition of the macroblock into the block shapes used for motion-compensated prediction. Partitions with luma block sizes of 16×16, 16×8, 8×16, and 8×8 samples are supported by the syntax. In case partitions with 8×8 samples are chosen, one additional syntax element for each 8×8 partition is transmitted. This syntax element specifies whether the corresponding 8×8 partition is further partitioned into partitions of 8×4, 4×8, or 4×4 luma samples and corresponding chroma samples. Fig. 12 illustrates the partitioning.

The prediction signal for each predictive-coded M×N luma block is obtained by displacing an area of the corresponding reference picture, which is specified by a translational motion vector and a picture reference index. Thus, if the macroblock is coded using four 8×8 partitions and each 8×8 partition is further split into four 4×4 partitions, a maximum of 16 motion vectors may be transmitted for a single P macroblock.

The accuracy of motion compensation is in units of one quarter of the distance between luma samples. In case the motion vector points to an integer-sample position, the prediction signal consists of the corresponding samples of the reference picture; otherwise the corresponding sample is obtained using interpolation to generate noninteger positions. The prediction values at half-sample positions are obtained by applying a one-dimensional 6-tap FIR filter horizontally and vertically. Prediction values at quarter-sample positions are generated by averaging samples at integer- and half-sample positions.

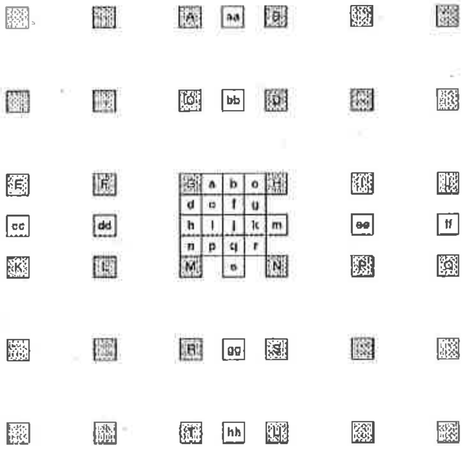


Fig. 13. Filtering for fractional-sample accurate motion compensation. Upper-case letters indicate samples on the full-sample grid, while lower case samples indicate samples in between at fractional-sample positions.

Fig. 13 illustrates the fractional sample interpolation for samples a–k and n–r. The samples at half sample positions labeled b and h are derived by first calculating intermediate values b_1 and h_1, respectively by applying the 6-tap filter as follows:

$$\begin{aligned} b_1 &= (E - 5F + 20G + 20H - 5I + J) \\ h_1 &= (A - 5C + 20G + 20M - 5R + T). \end{aligned}$$

The final prediction values for locations b and h are obtained as follows and clipped to the range of 0–255:

$$\begin{aligned} b &= (b_1 + 16) \gg 5 \\ h &= (h_1 + 16) \gg 5. \end{aligned}$$

The samples at half sample positions labeled as j are obtained by

$$j_1 = cc - 5dd + 20h_1 + 20m_1 - 5ee + ff$$

where intermediate values denoted as cc, dd, ee, m_1, and ff are obtained in a manner similar to h_1. The final prediction value j is then computed as $j = (j_1 + 512) \gg 10$ and is clipped to the range of 0 to 255. The two alternative methods of obtaining the value of j illustrate that the filtering operation is truly separable for the generation of the half-sample positions.

The samples at quarter sample positions labeled as a, c, d, n, f, i, k, and q are derived by averaging with upward rounding of the two nearest samples at integer and half sample positions as, for example, by

$$a = (G + b + 1) \gg 1.$$

The samples at quarter sample positions labeled as e, g, p, and r are derived by averaging with upward rounding of the two nearest samples at half sample positions in the diagonal direction as, for example, by

$$e = (b + h + 1) \gg 1.$$

The prediction values for the chroma component are always obtained by bilinear interpolation. Since the sampling grid of chroma has lower resolution than the sampling grid of the luma, the displacements used for chroma have one-eighth sample position accuracy.

The more accurate motion prediction using full sample, half sample and one-quarter sample prediction represent one of the major improvements of the present method compared to earlier standards for the following two reasons.

1) The most obvious reason is more accurate motion representation.
2) The other reason is more flexibility in prediction filtering. Full sample, half sample and one-quarter sample prediction represent different degrees of low pass filtering which is chosen automatically in the motion estimation process. In this respect, the 6-tap filter turns out to be a much better tradeoff between necessary prediction loop filtering and has the ability to preserve high-frequency content in the prediction loop.

A more detailed investigation of fractional sample accuracy is presented in [8].

The syntax allows so-called motion vectors over picture boundaries, i.e., motion vectors that point outside the image area. In this case, the reference frame is extrapolated beyond the image boundaries by repeating the edge samples before interpolation.

The motion vector components are differentially coded using either median or directional prediction from neighboring blocks. No motion vector component prediction (or any other form of prediction) takes place across slice boundaries.

The syntax supports multipicture motion-compensated prediction [9], [10]. That is, more than one prior coded picture can be used as reference for motion-compensated prediction. Fig. 14 illustrates the concept.

Multiframe motion-compensated prediction requires both encoder and decoder to store the reference pictures used for inter prediction in a multipicture buffer. The decoder replicates the multipicture buffer of the encoder according to memory management control operations specified in the bitstream. Unless the size of the multipicture buffer is set to one picture, the index at which the reference picture is located inside the multipicture buffer has to be signalled. The reference index parameter is transmitted for each motion-compensated 16×16, 16×8, 8×16, or 8×8 luma block. Motion compensation for smaller regions than 8×8 use the same reference index for prediction of all blocks within the 8×8 region.

In addition to the motion-compensated macroblock modes described above, a P macroblock can also be coded in the so-called P_Skip type. For this coding type, neither a quantized prediction error signal, nor a motion vector or reference index parameter is transmitted. The reconstructed signal is obtained similar to the prediction signal of a P_16×16 macroblock type that references the picture which is located at index 0 in the multipicture buffer. The motion vector used for reconstructing the P_Skip macroblock is similar to the motion vector predictor for the 16×16 block. The useful effect of this definition of the P_Skip coding type is that large areas with no change or constant motion like slow panning can be represented with very few bits.

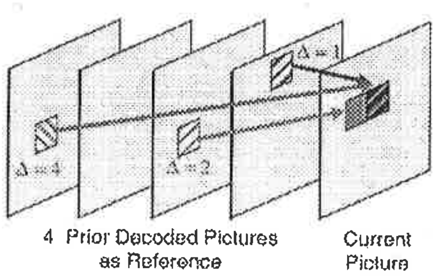


Fig. 14. Multiframe motion compensation. In addition to the motion vector, also picture reference parameters (Δ) are transmitted. The concept is also extended to B slices.

2) Inter-Frame Prediction in B Slices: In comparison to prior video coding standards, the concept of B slices is generalized in H.264/AVC. This extension refers back to [11] and is further investigated in [12]. For example, other pictures can reference pictures containing B slices for motion-compensated prediction, depending on the memory management control operation of the multipicture buffering. Thus, the substantial difference between B and P slices is that B slices are coded in a manner in which some macroblocks or blocks may use a weighted average of two distinct motion-compensated prediction values for building the prediction signal. B slices utilize two distinct lists of reference pictures, which are referred to as the first (list 0) and second (list 1) reference picture lists, respectively. Which pictures are actually located in each reference picture list is an issue of the multipicture buffer control and an operation very similar to the conventional MPEG-2 B pictures can be enabled if desired by the encoder.

In B slices, four different types of inter-picture prediction are supported: list 0, list 1, bi-predictive, and direct prediction. For the bi-predictive mode, the prediction signal is formed by a weighted average of motion-compensated list 0 and list 1 prediction signals. The direct prediction mode is inferred from previously transmitted syntax elements and can be either list 0 or list 1 prediction or bi-predictive.

B slices utilize a similar macroblock partitioning as P slices. Beside the P_16×16, P_16×8, P_8×16, P_8×8, and the intra coding types, bi-predictive prediction and another type of prediction called direct prediction, are provided. For each 16×16, 16×8, 8×16, and 8×8 partition, the prediction method (list 0, list 1, bi-predictive) can be chosen separately. An 8×8 partition of a B macroblock can also be coded in direct mode. If no prediction error signal is transmitted for a direct macroblock mode, it is also referred to as B_Skip mode and can be coded very efficiently similar to the P_Skip mode in P slices. The motion vector coding is similar to that of P slices with the appropriate modifications because neighboring blocks may be coded using different prediction modes.

I. Transform, Scaling, and Quantization

Similar to previous video coding standards, H.264/AVC utilizes transform coding of the prediction residual. However, in H.264/AVC, the transformation is applied to 4×4 blocks, and

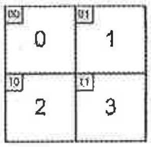


Fig. 15. Repeated transform for chroma blocks. The four blocks numbered 0–3 indicate the four chroma blocks of a chroma component of a macroblock.

instead of a 4×4 discrete cosine transform (DCT), a separable integer transform with similar properties as a 4×4 DCT is used. The transform matrix is given as

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}.$$

Since the inverse transform is defined by exact integer operations, inverse-transform mismatches are avoided. The basic transform coding process is very similar to that of previous standards. At the encoder, the process includes a forward transform, zig-zag scanning, scaling, and rounding as the quantization process followed by entropy coding. At the decoder, the inverse of the encoding process is performed except for the rounding. More details on the specific aspects of the transform in H.264/AVC can be found in [17].

It has already been mentioned that Intra_16×16 prediction modes and chroma intra modes are intended for coding of smooth areas. For that reason, the DC coefficients undergo a second transform with the result that we have transform coefficients covering the whole macroblock. An additional 2×2 transform is also applied to the DC coefficients of the four 4×4 blocks of each chroma component. The procedure for a chroma block is illustrated in Fig. 15. The small blocks inside the larger blocks represent DC coefficients of each of the four 4×4 chroma blocks of a chroma component of a macroblock numbered as 0, 1, 2, and 3. The two indices correspond to the indices of the 2×2 inverse Hadamard transform.

To explain the idea behind these repeated transforms, let us point to a general property of a two-dimensional transform of very smooth content (where sample correlation approaches 1). In that situation, the reconstruction accuracy is proportional to the inverse of the one-dimensional size of the transform. Hence, for a very smooth area, the reconstruction error with a transform covering the complete 8×8 block is halved compared to using only 4×4 transform. A similar rationale can be used for the second transform connected to the INTRA-16×16 mode.

There are several reasons for using a smaller size transform.

- One of the main improvements of the present standard is the improved prediction process both for inter and intra. Consequently, the residual signal has less spatial correlation. This generally means that the transform has less to offer concerning decorrelation. This also means that a 4×4 transform is essentially as efficient in removing statistical correlation as a larger transform
- With similar objective compression capability, the smaller 4×4 transform has visual benefits resulting in less noise around edges (referred to as "mosquito noise" or "ringing" artifacts).
- The smaller transform requires less computations and a smaller processing wordlength. Since the transformation process for H.264/AVC involves only adds and shifts, it is also specified such that mismatch between encoder and decoder is avoided (this has been a problem with earlier 8×8 DCT standards)

A quantization parameter is used for determining the quantization of transform coefficients in H.264/AVC. The parameter can take 52 values. Theses values are arranged so that an increase of 1 in quantization parameter means an increase of quantization step size by approximately 12% (an increase of 6 means an increase of quantization step size by exactly a factor of 2). It can be noticed that a change of step size by approximately 12% also means roughly a reduction of bit rate by approximately 12%.

The quantized transform coefficients of a block generally are scanned in a zig-zag fashion and transmitted using entropy coding methods. The 2×2 DC coefficients of the chroma component are scanned in raster-scan order. All inverse transform operations in H.264/AVC can be implemented using only additions and bit-shifting operations of 16-bit integer values. Similarly, only 16-bit memory accesses are needed for a good implementation of the forward transform and quantization process in the encoder.

J. Entropy Coding

In H.264/AVC, two methods of entropy coding are supported. The simpler entropy coding method uses a single infinite-extent codeword table for all syntax elements except the quantized transform coefficients. Thus, instead of designing a different VLC table for each syntax element, only the mapping to the single codeword table is customized according to the data statistics. The single codeword table chosen is an exp-Golomb code with very simple and regular decoding properties.

For transmitting the quantized transform coefficients, a more efficient method called Context-Adaptive Variable Length Coding (CAVLC) is employed. In this scheme, VLC tables for various syntax elements are switched depending on already transmitted syntax elements. Since the VLC tables are designed to match the corresponding conditioned statistics, the entropy coding performance is improved in comparison to schemes using a single VLC table.

In the CAVLC entropy coding method, the number of nonzero quantized coefficients (N) and the actual size and position of the coefficients are coded separately. After zig-zag scanning of transform coefficients, their statistical distribution typically shows large values for the low frequency part decreasing to small values later in the scan for the high-frequency part. An example for a typical zig-zag scan of quantized transform coefficients could be given as follows:

$$7, 6, \quad 2, 0, \quad 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0.$$

Based on this statistical behavior, the following data elements are used to convey information of quantized transform coefficients for a luma 4×4 block.

1) Number of Nonzero Coefficients (N) and "Trailing 1s": "Trailing 1s" (T1s) indicate the number of coefficients with absolute value equal to 1 at the end of the scan. In the example $T1s = 2$ and the number of coefficients is $N = 5$. These two values are coded as a combined event. One out of 4 VLC tables is used based on the number of coefficients in neighboring blocks.

2) Encoding the Value of Coefficients: The values of the coefficients are coded. The T1s need only sign specification since they all are equal to $+1$ or -1. Please note that the statistics of coefficient values has less spread for the last nonzero coefficients than for the first ones. For this reason, coefficient values are coded in reverse scan order. In the examples above, -2 is the first coefficient value to be coded. A starting VLC is used for that. When coding the next coefficient (having value of 6 in the example) a new VLC may be used based on the just coded coefficient. In this way adaptation is obtained in the use of VLC tables. Six exp-Golomb code tables are available for this adaptation.

3) Sign Information: One bit is used to signal coefficient sign. For T1s, this is sent as single bits. For the other coefficients, the sign bit is included in the exp-Golomb codes.

Positions of each nonzero coefficient are coded by specifying the positions of 0s before the last nonzero coefficient. It is split into two parts:

4) TotalZeroes: This codeword specifies the number of zeros between the last nonzero coefficient of the scan and its start. In the example the value of TotalZeros is 3. Since it is already known that $N = 5$, the number must be in the range 0–11. 15 tables are available for N in the range 1–15. (If $N = 16$ there is no zero coefficient.)

5) RunBefore: In the example it must be specified how the 3 zeros are distributed. First the number of 0s before the last coefficient is coded. In the example the number is 2. Since it must be in the range 0–3 a suitable VLC is used. Now there is only one 0 left. The number of 0s before the second last coefficient must therefore be 0 or 1. In the example the number is 1. At this point there are no 0s left and no more information is coded

The efficiency of entropy coding can be improved further if the Context-Adaptive Binary Arithmetic Coding (CABAC) is used [16]. On the one hand, the usage of arithmetic coding allows the assignment of a noninteger number of bits to each symbol of an alphabet, which is extremely beneficial for symbol probabilities that are greater than 0.5. On the other hand, the usage of adaptive codes permits adaptation to nonstationary symbol statistics. Another important property of CABAC is its context modeling. The statistics of already coded syntax elements are used to estimate conditional probabilities. These conditional probabilities are used for switching several estimated probability models. In H.264/AVC, the arithmetic coding core engine and its associated probability estimation are specified as multiplication-free low-complexity methods using only shifts and table look-ups. Compared to CAVLC, CABAC typically provides a reduction in bit rate between 5%–15% The highest gains are typically obtained when coding interlaced TV signals. More details on CABAC can be found in [16].

K. In-Loop Deblocking Filter

One particular characteristic of block-based coding is the accidental production of visible block structures. Block edges are typically reconstructed with less accuracy than interior pixels and "blocking" is generally considered to be one of the most visible artifacts with the present compression methods. For this reason, H.264/AVC defines an adaptive in-loop deblocking filter, where the strength of filtering is controlled by the values of several syntax elements. A detailed description of the adaptive deblocking filter can be found in [18].

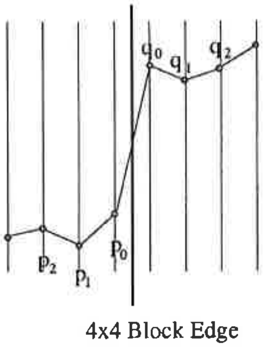


Fig. 16. Principle of deblocking filter.

Fig. 16 illustrates the principle of the deblocking filter using a visualization of a one-dimensional edge. Whether the samples p_0 and q_0 as well as p_1 and q_1 are filtered is determined using quantization parameter (QP) dependent thresholds $\alpha(\mathrm{QP})$ and $\beta(\mathrm{QP})$. Thus, filtering of p_0 *and* q_0 only takes place if each of the following conditions is satisfied:

$$1.\ |p_0 - q_0| < \alpha(\mathrm{QP})$$
$$2.\ |p_1 - p_0| < \beta(\mathrm{QP})$$
$$3.\ |q_1 - q_0| < \beta(\mathrm{QP})$$

where the $\beta(\mathrm{QP})$ is considerably smaller than $\alpha(\mathrm{QP})$. Accordingly, filtering of p_1 or q_1 takes place if the corresponding following condition is satisfied:

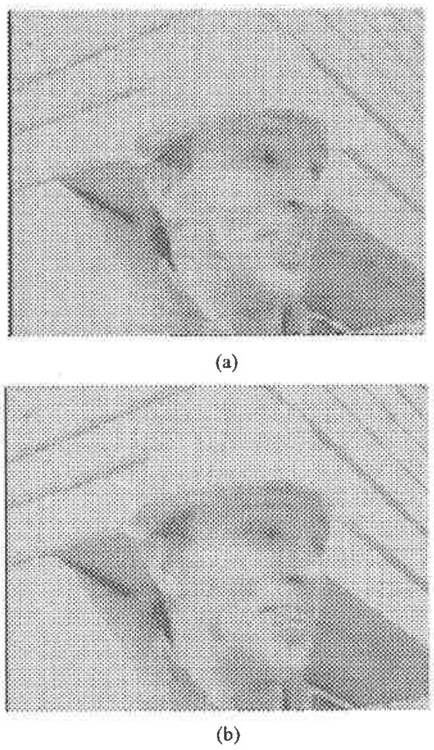
$$|p_2 - p_0| < \beta(\mathrm{QP}) \text{ or } |q_2 q_0| < \beta(\mathrm{QP}).$$

The basic idea is that if a relatively large absolute difference between samples near a block edge is measured, it is quite likely a blocking artifact and should therefore be reduced. However, if the magnitude of that difference is so large that it cannot be explained by the coarseness of the quantization used in the encoding, the edge is more likely to reflect the actual behavior of the source picture and should not be smoothed over.

The blockiness is reduced, while the sharpness of the content is basically unchanged. Consequently, the subjective quality is significantly improved. The filter reduces the bit rate typically by 5%–10% while producing the same objective quality as the nonfiltered video. Fig. 17 illustrates the performance of the deblocking filter.

L. Hypothetical Reference Decoder

One of the key benefits provided by a standard is the assurance that all the decoders compliant with the standard will be able to decode a compliant compressed video. To achieve that, it is not sufficient to just provide a description of the coding algorithm. It is also important in a real-time system to specify how bits are fed to a decoder and how the decoded pictures


(a)

(b)

Fig. 17. Performance of the deblocking filter for highly compressed pictures (a) without deblocking filter and (b) with deblocking filter.

are removed from a decoder. Specifying input and output buffer models and developing an implementation independent model of a receiver achieves this. That receiver model is also called hypothetical reference decoder (HRD) and is described in detail in [19]. An encoder is not allowed to create a bitstream that cannot be decoded by the HRD. Hence, if in any receiver implementation the designer mimics the behavior of HRD, it is guaranteed to be able to decode all the compliant bitstreams.

In H.264/AVC HRD specifies operation of two buffers: 1) the coded picture buffer (CPB) and 2) the decoded picture buffer (DPB). CPB models the arrival and removal time of the coded bits. The HRD design is similar in spirit to what MPEG-2 had, but is more flexible in support of sending the video at a variety of bit rates without excessive delay.

Unlike MPEG-2, in H.264/AVC, multiple frames can be used for reference, the reference frames can be located either in past or future arbitrarily in display order, the HRD also specifies a model of the decoded picture buffer management to ensure that excessive memory capacity is not needed in a decoder to store the pictures used as references.

V. Profiles and Potential Applications

A. Profiles and Levels

Profiles and levels specify conformance points. These conformance points are designed to facilitate interoperability between various applications of the standard that have similar functional requirements. A *profile* defines a set of coding tools or algorithms that can be used in generating a conforming bitstream, whereas a *level* places constraints on certain key parameters of the bitstream.

All decoders conforming to a specific profile must support all features in that profile. Encoders are not required to make use of any particular set of features supported in a profile but have to provide conforming bitstreams, i.e., bitstreams that can be decoded by conforming decoders. In H.264/AVC, three profiles are defined, which are the Baseline, Main, and Extended Profile.

The Baseline profile supports all features in H.264/AVC except the following two feature sets:

- **Set 1:** B slices, weighted prediction, CABAC, field coding, and picture or macroblock adaptive switching between frame and field coding.
- **Set 2:** SP/SI slices, and slice data partitioning.

The first set of additional features is supported by the Main profile. However, the Main profile does not support the FMO, ASO, and redundant pictures features which are supported by the Baseline profile. Thus, only a subset of the coded video sequences that are decodable by a Baseline profile decoder can be decoded by a Main profile decoder. (Flags are used in the sequence parameter set to indicate which profiles of decoder can decode the coded video sequence).

The Extended Profile supports all features of the Baseline profile, and both sets of features on top of Baseline profile, except for CABAC.

In H.264/AVC, the same set of level definitions is used with all profiles, but individual implementations may support a different level for each supported profile. There are 15 levels defined, specifying upper limits for the picture size (in macroblocks) ranging from QCIF to all the way to above 4k×2k, decoder-processing rate (in macroblocks per second) ranging from 250k pixels/s to 250M pixels/s, size of the multipicture buffers, video bit rate ranging from 64 kbps to 240 Mbps, and video buffer size.

B. Areas for the Profiles of the New Standard to be Used

The increased compression efficiency of H.264/AVC offers to enhance existing applications or enables new applications. A list of possible application areas is provided below.

- Conversational services which operate typically below 1 Mbps with low latency requirements. The ITU-T SG16 is currently modifying its systems recommendations to support H.264/AVC use in such applications, and the IETF is working on the design of an RTP payload packetization. In the near term, these services would probably utilize the Baseline profile (possibly progressing over time to also use other profiles such as the Extended profile). Some specific applications in this category are given below.
  —H.320 conversational video services that utilize circuit-switched ISDN-based video conferencing.
  —3GPP conversational H.324/M services.
  —H.323 conversational services over the Internet with best effort IP/RTP protocols.
  —3GPP conversational services using IP/RTP for transport and SIP for session setup.
- Entertainment video applications which operate between 1–8+ Mbps with moderate latency such as 0.5 to 2 s.

The H.222.0/MPEG-2 Systems specification is being modified to support these application. These applications would probably utilize the Main profile and include the following.

—Broadcast via satellite, cable, terrestrial, or DSL.
—DVD for standard and high-definition video.
—Video-on-demand via various channels.

- Streaming services which typically operate at 50–1500 kbps and have 2 s or more of latency. These services would probably utilize the Baseline or Extended profile and may be distinguished by whether they are used in wired or wireless environments as follows:

  —3GPP streaming using IP/RTP for transport and RTSP for session setup. This extension of the 3GPP specification would likely use Baseline profile only.

  —Streaming over the wired Internet using IP/RTP protocol and RTSP for session setup. This domain which is currently dominated by powerful proprietary solutions might use the Extended profile and may require integration with some future system designs.

- Other services that operate at lower bit rates and are distributed via file transfer and therefore do not impose delay constraints at all, which can potentially be served by any of the three profiles depending on various other systems requirements are:

  —3GPP multimedia messaging services;
  —video mail.

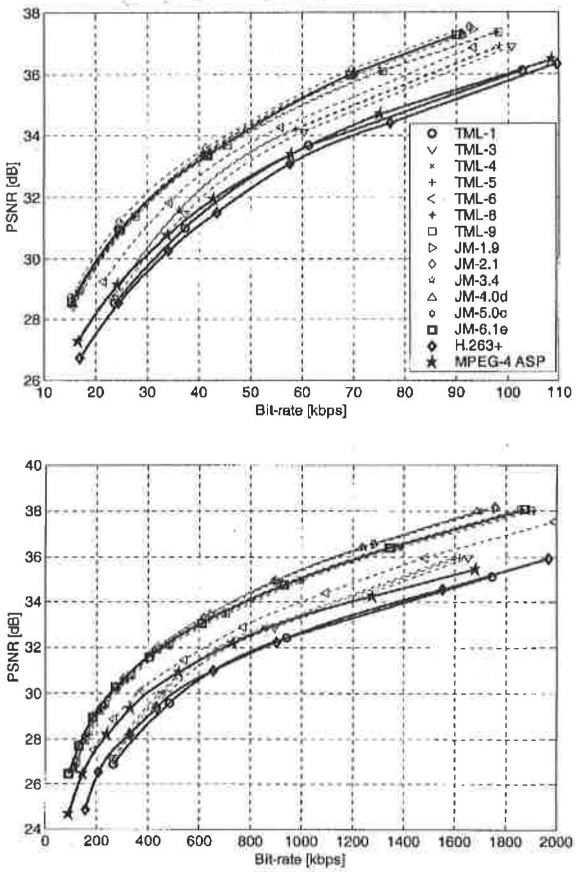


Fig. 18. Evolution of H.264/AVC since August 1999 until March 2003. Top: QCIF sequence Foreman coded at 10 Hz. Bottom: CIF sequence tempete coded at 30 Hz. The legend in the top figure indicates the various versions that have been run with typical settings.

VI. History and Standardization Process

In this section, we illustrate the history of the standard. The development of H.264/AVC is characterized by improvements in small steps over the last 3–4 years as can be seen from Fig. 18. In Fig. 18, the coding performance is shown for two example progressive-scan video sequences, when enabling the typical coding options for the various versions of the standard since August 1999 until completion in April 2003. The dates and creation of the various versions are shown in Table I. The document and the software versions have been called test model long-term (TML) when being developed in VCEG and joint model (JM) when the development was continued in the joint video team (JVT) as a partnership between MPEG and VCEG. The development took place in small steps between each version of the design as can be seen from Fig. 18.

The work started in VCEG as a parallel activity to the completion of the last version of H.263. The first Test Model Long-Term (TML-1, curve with circles in Fig. 18) was produced in August 1999. TML-1 was similar to H.263 by using a combination of block prediction and block transform/quantization/coding of the residual signal. The PSNR performance of TML-1 was similar to that of H.263 (curve with diamond-shaped markers in Fig. 18) and below MPEG-4 ASP (curve with star-shaped markers in Fig. 18). However, the starting point was considered sufficient due to some perceptual benefits being shown and a judgment that the design could be incrementally improved. The results shown for H.263 and MPEG-4 ASP have been optimized using Lagrangian methods as described in [13]. The main coding features of TML-1 are summarized as follows:

- Seven block partitions for inter prediction of a macroblock. The luminance component of a macroblock could be partitioned into 16×16, 16×8, 8×16, 8×8, 8×4, 4×8 or 4×4 blocks in a similar way as depicted in Fig. 12. The 16×16, 16×8, 8×16, and 8×8 blocks have remained in the design. Partitioning into smaller blocks was modified later (in JM-2.1) into a tree-structured macroblock partition as described above.
- 1/3-sample accurate motion-compensated prediction using 4-tap filter in horizontal and vertical direction. The filter taps were (−1, 12, 6, −1)/16 and (−1, 6, 12, −1)/16. One of the sample positions used a stronger low pass filter for more flexible prediction loop filtering. This was modified later (in TML-4) to 1/4-sample accurate prediction using a 6-tap as described above. For TML-2, a method called adaptive motion accuracy (AMA) was adopted which has never been implemented into the software. AMA was dropped due to lack of coding efficiency improvement in TML-4. In TML-7/8, 1/8-sample accurate motion compensation was introduced which was then dropped for complexity reasons in JM-5.

TABLE I
HISTORY OF H.264/AVC STANDARDIZATION PROCESS. FOR TML VERSIONS WITH A *, NO SOFTWARE HAS BEEN CREATED

No	TML/JM	Date	Location
1	TML-1	Aug. 1999	Berlin, Germany
2	TML-2*	Oct. 1999	Red Bank, NJ, USA
3	TML-3	Feb. 2000	Geneva, Switzerland
4	TML-4	May, 2000	Osaka, Japan
5	TML-5	Aug. 2000	Portland, OR, USA
6	TML-6	Jan. 2001	Eibsee, Germany
7	TML-7*	Apr. 2001	Austin, TX, USA
8	TML-8	May 2001	Porto Seguro, Brazil
9	TML-9	Sep. 2001	Santa Barbara, CA, USA
10	JM-1	Dec. 2001	Pattaya, Thailand
11	JM-2	Feb. 2002	Geneva, Switzerland
12	JM-3	May 2002	Fairfax, VA, USA
13	JM-4	July 2002	Klagenfurt, Austria
14	JM-5	Oct. 2002	Geneva, Switzerland
15	JM-6	Dec. 2002	Awaji, Japan
16	Final	Mar. 2003	Pattaya, Thailand

- Multiple reference frames, the decoupling of temporal order and display order, and the decoupling of picture type from the ability to use pictures as references were envisioned from the beginning, but were integrated in the software and draft text rather gradually.
- Intra prediction was done on 4×4 blocks and based on the neighboring samples. There were five prediction modes which were the ones shown in Fig. 9 except with a simpler version of Mode 3. The number of prediction modes was increased to 7 in TML-4 and further increased to 9 in JM-2. In TML-3, the Intra_16×16 prediction mode was introduced and in JM-3, the various prediction modes for chroma have been introduced.
- The transform for the residual signal had size 4×4. This was in contrast to all previous standards which used transform size 8×8. The transform was also no longer exact DCT but an integer transform very close to DCT. The integer definition resulted in an exact definition of the inverse transform. The transform had basis vectors

$$H = \begin{bmatrix} 13 & 13 & 13 & 13 \\ 17 & 7 & -7 & 17 \\ 13 & -13 & -13 & 13 \\ 7 & -17 & 17 & -7 \end{bmatrix}.$$

  The transform was later changed to the version described above in JM-2.
- An in-loop deblocking filter similar to the one used in H.263 was in TML-1, but only on intra frames. This filter has been considerably refined during the entire development of the standard.
- Entropy coding used one single exp-Golomb type VLC table for all syntax elements (including transform coefficients). This was extended later by CABAC in TML-7 and CAVLC for transform coefficients in JM-3.

TML-1 did not contain many of the features of the final design of JM-6 (squares in Fig. 18) including interlace support, B pictures and the NAL. The method of handling of interlaced video was among the last things integrated into the design (note that Fig. 18 does not show performance for interlaced video). The improvement of JM-6 relative to TML-1 is typically between 2–3 dB PSNR or between 40%–60% in bit-rate reduction.

VII. CONCLUSIONS

The emerging H.264/AVC video coding standard has been developed and standardized collaboratively by both the ITU-T VCEG and ISO/IEC MPEG organizations. H.264/AVC represents a number of advances in standard video coding technology, in terms of both coding efficiency enhancement and flexibility for effective use over a broad variety of network types and application domains. Its VCL design is based on conventional block-based motion-compensated hybrid video coding concepts, but with some important differences relative to prior standards. We thus summarize some of the important differences:

- enhanced motion-prediction capability;
- use of a small block-size exact-match transform;
- adaptive in-loop deblocking filter;
- enhanced entropy coding methods.

When used well together, the features of the new design provide approximately a 50% bit rate savings for equivalent perceptual quality relative to the performance of prior standards (especially for higher-latency applications which allow some use of reverse temporal prediction).[1]

ACKNOWLEDGMENT

The authors thank the experts of ITU-T VCEG, ISO/IEC MPEG, and the ITU-T/ISO/IEC Joint Video Team for their contributions. Thanks to K. Sühring and H. Schwarz for providing the data for the comparison of the various TML and JM versions.

REFERENCES

[1] "Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T Rec. H.264/ISO/IEC 14 496-10 AVC," in Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, JVT-G050, 2003.

[2] "Generic Coding of Moving Pictures and Associated Audio Information - Part 2: Video," ITU-T and ISO/IEC JTC 1, ITU-T Recommendation H.262 and ISO/IEC 13 818-2 (MPEG-2), 1994.

[3] "Video Codec for Audiovisual Services at $p \times 64$ kbit/s ITU-T Recommendation H.261, Version 1," ITU-T, ITU-T Recommendation H.261 Version 1, 1990.

[4] "Video Coding for Low Bit Rate Communication," ITU-T, ITU-T Recommendation H.263 version 1, 1995.

[5] "Coding of audio-visual objects—Part 2: Visual," in *ISO/IEC 14 496-2 (MPEG-4 Visual Version 1)*, Apr. 1999.

[6] S. Wenger, "H.264/AVC over IP," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 645–656, July 2003.

[7] T. Stockhammer, M. M. Hannuksela, and T. Wiegand, "H.264/AVC in wireless environments," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 657–673, July 2003.

[1] Further information and documents of the project is available by ftp at ftp://ftp.imtc-files.org/jvt-experts.

[8] T. Wedi, "Motion compensation in H.264/AVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 577–586, July 2003.

[9] T. Wiegand, X. Zhang, and B. Girod, "Long-term memory motion-compensated prediction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, pp. 70–84, Feb. 1999.

[10] T. Wiegand and B. Girod, *Multi-Frame Motion-Compensated Prediction for Video Transmission*. Norwell, MA: Kluwer, 2001.

[11] M. Flierl, T. Wiegand, and B. Girod, "A locally optimal design algorithm for block-based multi-hypothesis motion-compensated prediction," in *Proc. Data Compression Conf.*, Snowbird, UT, Mar. 1998, pp. 239–248.

[12] M. Flierl and B. Girod, "Generalized B pictures and the draft JVT/H.264 video compression standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 587–597, July 2003.

[13] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G. J. Sullivan, "Rate-constrained coder control and comparison of video coding standards," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 688–703, July 2003.

[14] S. Wenger, "H.264/AVC over IP," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 645–656, July 2003.

[15] M. Karczewicz and R. Kurçeren, "The SP and SI frames design for H.264/AVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 637–644, July 2003.

[16] D. Marpe, H. Schwarz, and T. Wiegand, "Context-adaptive binary arithmetic coding in the H.264/AVC video compression standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 620–636, July 2003.

[17] H. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-Complexity transform and quantization in H.264/AVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 598–603, July 2003.

[18] P. List, A. Joch, J. Lainema, G. Bjøntegaard, and M. Karczewicz, "Adaptive deblocking filter," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 614–619, July 2003.

[19] J. Ribas-Corbera, P. A. Chou, and S. Regunathan, "A generalized hypothetical reference decoder for H.264/AVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 674–687, July 2003.

Thomas Wiegand received the Dr.-Ing. degree from the University of Erlangen-Nuremberg, Germany, in 2000 and the Dipl.-Ing. degree in electrical engineering from the Technical University of Hamburg-Harburg, Germany, in 1995.

He is the Head of the Image Communication Group in the Image Processing Department, Fraunhofer–Institute for Telecommunications – Heinrich Hertz Institute (HHI), Berlin, Germany. During 1997 to 1998, he was a Visiting Researcher at Stanford University, Stanford, CA, and served as a Consultant to 8x8, Inc., Santa Clara, CA. From 1993 to 1994, he was a Visiting Researcher at Kobe University, Kobe, Japan. In 1995, he was a Visiting Scholar at the University of California at Santa Barbara, where he began his research on video compression and transmission. Since then, he has published several conference and journal papers on the subject and has contributed successfully to the ITU-T Video Coding Experts Group (ITU-T SG16 Q.6—VCEG)/ISO/IEC Moving Pictures Experts Group (ISO/IEC JTC1/SC29/WG11—MPEG)/Joint Video Team (JVT) standardization efforts and holds various international patents in this field. He has been appointed as the Associated Rapporteur of the ITU-T VCEG (October 2000), the Associated Rapporteur/Co-Chair of the JVT that has been created by ITU-T VCEG and ISO/IEC MPEG for finalization of the H.264/AVC video coding standard (December 2001), and the Editor of the H.264/AVC video coding standard (February 2002).

Gary J. Sullivan (S'83–M' 91–SM'01) received the B.S. and M.Eng. degrees in electrical engineering from the University of Louisville, Louisville, KY, in 1982 and 1983, respectively, and the Ph.D. and Eng. degrees in electrical engineering from the University of California, Los Angeles, in 1991.

He is the Chairman of the Joint Video Team (JVT) for the development of the next-generation H.264/MPEG4-AVC video coding standard, which was recently completed as a result of a joint project between the ITU-T video coding experts group (VCEG) and the ISO/IEC moving picture experts group (MPEG). He is also the Rapporteur of Advanced Video Coding in the ITU-T, where he has led VCEG (ITU-T Q.6/SG16) for about six years, and the ITU-T Video Liaison Representative to MPEG (ISO/IEC JTC1/SC29/WG11) and served as MPEG's Video Chairman during 2001–2002. He is currently a Program Manager of video standards and technologies in the eHome A/V Platforms Group of Microsoft Corporation, Redmond, WA, where he designed and remains lead engineer for the DirectX® Video Acceleration API/DDI feature of the Microsoft Windows® operating system platform. Prior to joining Microsoft in 1999, he was the Manager of Communications Core Research at PictureTel Corporation, the quondam world leader in videoconferencing communication. He was previously a Howard Hughes Fellow and Member of the Technical Staff in the Advanced Systems Division of Hughes Aircraft Corporation and was a terrain-following radar system software engineer for Texas Instruments. His research interests and areas of publication include image and video compression, rate-distortion optimization, motion representation, scalar and vector quantization, and error- and packet-loss-resilient video coding.

Gisle Bjøntegaard received the Dr. Phil. degree in physics from the University of Oslo, Oslo, Norway, in 1974.

From 1974 to 1996, he was a Senior Scientist with Telenor Research and Development, Oslo, Norway. His areas of research included radio link network design, reflector antenna design and construction, digital signal procession, and development of video compression methods. From 1996 to 2002, he was a Group Manager at Telenor Broadband Services, Oslo, Norway, where his areas of work included the design of point-to-point satellite communication and development of satellite digital TV platform. Since 2002, he has been a Principal Scientist at Tandberg Telecom, Lysaker, Norway, working with video-coding development and implementation. He has contributed actively to the development of the ITU video standards H.261, H.262, H.263, and H.264, as well as to ISO/IEC MPEG2 and MPEG4.

Ajay Luthra (S'79–M'81–SM'89) received the B.E. (Hons.) degree from BITS, Pilani, India, in 1975, the M.Tech. degree in communications engineering from IIT Delhi, Delhi, India, in 1977, and the Ph.D. degree from Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, in 1981.

From 1981 to 1984, he was a Senior Engineer at Interspec Inc., Philadelphia, PA, where he was involved in applications of digital signal and image processing for Bio-medical applications. From 1984 to 1995, he was with Tektronix, Beaverton, OR, where he was Manager of the Digital Signal and Picture Processing Group during 1985–1990 and then Director of the Communications/Video Systems Research Lab from 1990–1995 . He is currently a Senior Director in Advanced Technology Group at Motorola, Broadband Communications Sector (formerly General Instrument), San Diego, CA, where he is involved in advanced development work in the areas of digital video compression and processing, streaming video, interactive TV, cable head-end system design, and advanced set-top box architectures. He has been an active member of the MPEG committee for the last ten years where he has chaired several technical subgroups. He is an Associate Rapporteur/Co-Chair of the Joint Video Team consisting of ISO/MPEG and ITU-T/H.26L experts working on developing next generation of video coding standards.

Dr. Luthra was an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY (2000–2002) and also a Guest Editor for its Special Issue on Streaming Video (March 2001).